REFORMA 2019

Część 2

Projektowanie, programowanie i testowanie aplikacji

INF.04



Podręcznik do nauki zawodu

TECHNIK PROGRAMISTA

WSiP

Autor: **Dariusz Gołębiowski**

Typ szkoły: **technikum**
Zawód: **technik programista**
Kwalifikacja: **INF.04. Projektowanie, programowanie i testowanie aplikacji**

© Copyright by Wydawnictwa Szkolne i Pedagogiczne
Warszawa 2021

Wydanie I (2021)

ISBN 978-83-02-19667-6 (część 2) (podręcznik w postaci elektronicznej – flipbook)
ISBN 978-83-02-19668-3 (całość) (podręcznik w postaci elektronicznej – flipbook)
ISBN 978-83-02-19660-7 (część 2) (podręcznik w postaci papierowej)
ISBN 978-83-02-19661-4 (całość) (podręcznik w postaci papierowej)

Opracowanie merytoryczne i redakcyjne: **Tomasz Kompanowski**, **Sebastian Jabłoński** (redakcja merytoryczna)
Redakcja językowa: **Elżbieta Michalak**
Redakcja techniczna: **Elżbieta Walczak**
Projekt okładki: **Dominik Krajewski**
Ilustracje: **Dominik Krajewski**
Skład i łamanie: **Ledor**

Wydawnictwa Szkolne i Pedagogiczne Spółka Akcyjna
00-807 Warszawa, Aleje Jerozolimskie 96
KRS 0000595068
Infolinia: 801 220 555
www.wsip.pl
Druk i oprawa: Drukarnia Interak Sp. z o.o., Czarnków

Publikacja, którą nabyłaś / nabyłeś, jest dziełem twórcy i wydawcy. Prosimy, abyś przestrzegała / przestrzegał praw, jakie im przysługują. Jej zawartość możesz udostępnić nieodpłatnie osobom bliskim lub osobiście znanym. Ale nie publikuj jej w internecie. Jeśli cytujesz jej fragmenty, nie zmieniaj ich treści i koniecznie zaznacz, czyje to dzieło. A kopiując jej część, rób to jedynie na użytek osobisty.

prawolubni

Szanujmy cudzą własność i prawo.
Więcej na www.legalnakultura.pl
Polska Izba Książki

I. Projektowanie oprogramowania ... 7

1. Algorytmy sortowania i wyszukiwania ... 8
   - 1.1. Algorytmy sortowania ... 8
   - 1.2. Algorytmy wyszukiwania ... 13
2. Wzorce projektowe ... 17
   - 2.1. Wzorce projektowe ... 17
   - 2.2. Fasada ... 17
   - 2.3. Metoda szablonowa ... 19
3. Prawo autorskie w dziedzinie programowania ... 24
   - 3.1. Autorskie prawa osobiste i majątkowe ... 24
   - 3.2. Czas trwania praw autorskich ... 27
   - 3.3. Konsekwencje naruszenia prawa autorskiego ... 28
   - 3.4. Elementy własności intelektualnej (dobra niematerialne, własności przemysłowe) ... 29
   - 3.5. Typy licencji oprogramowania ... 31

II. Programowanie obiektowe ... 37

4. Środowisko programistyczne dla obiektowych aplikacji konsolowych ... 38
   - 4.1. Środowisko programistyczne – wprowadzenie ... 38
   - 4.2. Kompilator, interpreter, debugger ... 48
   - 4.3. Etapy kompilacji i interpretacji kodu ... 52
   - 4.4. Biblioteki i frameworki ... 56
   - 4.5. Kompilowanie i uruchamianie programów ... 62
5. Podstawowe zasady programowania ... 66
   - 5.1. Funkcje – kody do wielokrotnego stosowania ... 66
   - 5.2. Rekurencja ... 71
   - 5.3. Programowanie jako zapis algorytmów ... 73
6. Typy danych w programowaniu na bazie projektu Gra edukacyjna Zgadnij Liczbę ... 77
   - 6.1. Proste i złożone typy danych ... 77
   - 6.2. Projekt Gra edukacyjna Zgadnij Liczbę – deklarowanie zmiennych różnych typów danych ... 80
   - 6.3. Projekt Gra edukacyjna Zgadnij Liczbę – operacje na zmiennych wejścia i wyjścia ... 82
   - 6.4. Projekt Gra edukacyjna Zgadnij Liczbę – typy złożone i operacje na nich ... 83
   - 6.5. Deklarowanie własnych typów danych ... 90
7. Ulepszenie projektu Gra edukacyjna Zgadnij Liczbę – wyrażenia, instrukcje i biblioteki ... 92
   - 7.1. Podsumowanie z odrobiną teorii i drobnymi ulepszeniami ... 92
   - 7.2. Sprawdzenie typu – czy zmienna jest typu int ... 94
   - 7.3. Dołączenie pierwszej biblioteki do tworzonego projektu ... 97
8. Zasady programowania obiektowego ... 99
   - 8.1. Świat obiektów ... 99
   - 8.2. Podstawowe pojęcia z zakresu programowania obiektowego ... 101
   - 8.3. Hermetyzacja, dziedziczenie i polimorfizm ... 103
   - 8.4. Tworzenie klas i obiektów ... 106
9. Klasy i ich wykorzystanie w programowaniu ... 110
   - 9.1. Zakres widoczności metod, pól klasy – kwalifikatory dostępu ... 110
   - 9.2. Deklaracja klasy, konstruktory i destruktory ... 114
   - 9.3. Tworzenie obiektów na bazie klasy ... 116
   - 9.4. Klasy dziedziczone i zaprzyjaźnione ... 118
   - 9.5. Składnik statyczny klasy i metody ich obsługi ... 123

10. Klasy pochodne i wzorce klas 126
10.1. Hierarchia dziedziczenia klas w programie 126
10.2. Klasy bazowe i pochodne 128
10.3. Metody wirtualne i abstrakcyjne klasy 130
11. Programowanie wyjątków 134
11.1. Instrukcje try i catch 134
11.2. Instrukcja throw 135
11.3. Obsługa błędów wykonania aplikacji 136

III. Programowanie aplikacji desktopowych 141

12. Środowisko programistyczne dla aplikacji desktopowych 142
12.1. Środowisko IDE 142
12.2. Środowisko RAD 144
12.3. Framework 146
13. Wykorzystywanie frameworków do programowania aplikacji desktopowych 148
13.1. Elementy interfejsu użytkownika 148
13.2. Okno aplikacji desktopowej 149
13.3. Tekst 151
13.4. Przyciski 155
13.5. Wejście użytkownika 158
13.6. Zaawansowane wejście użytkownika, obsługa klawiatury i zdarzenia myszy 163
14. Programowanie desktopowych aplikacji okienkowych za pomocą języka programowania Python 172
14.1. Planowanie aplikacji 172
14.2. Pisanie aplikacji na bazie poznanych wcześniej elementów interfejsu użytkownika 176
14.3. Eksport aplikacji do pliku .exe 187

IV. Programowanie aplikacji mobilnych 193

15. Wykorzystanie środowiska programistycznego dla aplikacji mobilnych na podstawie języka Python. 194
15.1. Wprowadzenie 194
15.2. Stworzenie środowiska wirtualnego 197
15.3. Pierwsza aplikacja mobilna 201
15.4. Przygotowanie pakietów 203
15.5. Alternatywne technologie tworzenia aplikacji mobilnych 209
16. Programowanie aplikacji mobilnych opartych na języku programowania Python 217
16.1. Kivy – wprowadzenie do frameworka 217
16.2. UI, GUI, WUI – wyjaśnienie podstawowych pojęć 218
16.3. Elementy UI aplikacji mobilnej 221
16.4. Formularz – projekt aplikacji opartej na GridLayout i Pythonie 226
16.5. Język kv – projektowanie graficzne aplikacji 232
16.6. Projekt aplikacji – Stoper 241
16.7. Współpraca Kivy z relacyjną bazą danych SQL 247
16.8. Ekrany aplikacji mobilnej 265
16.9. Omówienie przygotowania aplikacji do publikacji w sklepie – podsumowanie 273

V. Testowanie i dokumentowanie aplikacji 285

17. Walidacja kodu programu 286
17.1. Weryfikacja projektu informatycznego – podstawowe pojęcia 286
17.2. Narzędzia i środowisko do testowania programów 291
17.3. Walidacja i obsługa błędów 295
18. Tworzenie dokumentacji podczas programowania aplikacji 298
18.1. Dokumentowanie w kodzie źródłowym programu 298
18.2. Dokumentacja opisowa kodów programu 302
18.3. Instrukcja użytkowania aplikacji 302
19. Testowanie aplikacji 304
19.1. Testowanie w informatyce – podstawy wiedzy 304
19.2. Testy funkcjonalne i niefunkcjonalne 305
19.3. Testowanie aplikacji (w tym interfejsu) 309
19.4. Proces testowania i obsługi błędów w języku programowania Python 310

Dodatek 320

Netografia 335

Wykaz podstawowych pojęć w językach: polskim, angielskim i niemieckim 339

I. Projektowanie oprogramowania

- Algorytmy sortowania i wyszukiwania
- Wzorce projektowe
- Prawo autorskie w dziedzinie programowania

7 Algorytmy sortowania i wyszukiwania

ZAGADNIENIA

- Algorytmy sortowania
- Algorytmy wyszukiwania

1.1. Algorytmy sortowania

Jednym z elementów pracy programisty i pisanych przez niego programów jest zarządzanie ogromnymi ilościami informacji. Mając do czynienia z różnymi danymi, warto je umiejętnie posortować. Istnieje wiele metod sortowania – od bardzo prostych po zaawansowane.

Najbardziej znane algorytmy sortowania to:

- bąbelkowe,
- przez wstawianie,
- szybkie (quicksort),
- przez scalanie.

Sortowanie bąbelkowe to najprostszy sposób sortowania o złożoności czasowej O(n^2). Algorytm polega na porównywaniu po kolei par elementów tablicy, zaczynając od lewej strony, i zamianie elementu, jeżeli ten po lewej jest większy; w przeciwnym razie pomijamy element po lewej i idziemy w prawo, algorytm powtarzamy n-razy (gdzie n to długość naszej tablicy). Algorytm nazywa się bąbelkowy, gdyż elementy „lżejsze" idą do góry (w lewo), a „cięższe" opadają (w prawo).

Sortowanie bąbelkowe, tak jak każde inne rozwiązanie, ma zalety i wady.

Zalety:

- bardzo prosta konstrukcja informatyczna, umożliwiająca utworzenie implementacji na każdym typie prostym;
- do zamiany sortowania z kolejności rosnącej na malejącą wystarczy zmienić znak większy od „>" na znak mniejszy od „<".

Wadą zaś jest to, że z powodu dużej złożoności obliczeniowej algorytm nie wykona zamiany elementów w części iteracji.

```python
def bubble_sort(tablica):
    for j in range(len(tablica)):
        for i in range(0, len(tablica)-1):
            if tablica[i] > tablica[i+1]:
                temp = tablica[i]
                tablica[i] = tablica[i+1]
                tablica[i+1] = temp
    return tablica

tablica = [3, 7, 1, 8, 2, 9, 6]
# w pythonie, aby nie nadpisać tablicy, używamy jej kopii
# z użyciem metody .copy()
tabs = bubble_sort(tablica.copy())
print(tablica)
print(tabs)
```

Rys. 1.1. Kod języka programowania Python – algorytm sortowania bąbelkowego

```
PS D:\SOAW2> python test.py
[3, 7, 1, 8, 2, 9, 6]
[1, 2, 3, 6, 7, 8, 9]
```

Rys. 1.2. Wynik działania programu – algorytm sortowania bąbelkowego

Sortowanie przez wstawianie służy do sortowania o złożoności czasowej O(n^2). Algorytm polega na wybraniu po kolei elementów i porównywaniu ich z resztą oraz przestawianiu, aż nie dostaniemy posortowanej tablicy. Dokładniej wygląda on tak:

1. Wybieramy element pierwszy po lewej i porównujemy go z elementem po prawej. Jeżeli jest większy, to zamieniamy elementy posortowane.
2. Wybieramy kolejny element z nieposortowanej części tablicy i porównujemy go z częścią posortowaną i wstawiamy w odpowiednie miejsce.
3. Wykonujemy punkt 2. tyle razy, aż tablica będzie posortowana.

```python
def sortowanie_przez_wstawianie(tablica):
    posortowane = 1
    tablica = list(tablica)
    for i in range(1, len(tablica)):
        for j in range(posortowane):
            if tablica[i] < tablica[j]:
                tablica.insert(j, tablica.pop(i))
                break
        posortowane += 1
    return tablica


tablica = [3, 7, 1, 8, 2, 9, 6]
tabs = sortowanie_przez_wstawianie(tablica.copy())
print(tablica)
print(tabs)
```

Rys. 1.3. Kod języka programowania Python – algorytm sortowania przez wstawianie

```
PS D:\SOAW2> python test.py
[3, 7, 1, 8, 2, 9, 6]
[1, 2, 3, 6, 7, 8, 9]
```

Rys. 1.4. Wynik działania programu – algorytm sortowania przez wstawianie

Sortowanie szybkie (**quicksort**) to jeden z najszybszych algorytmów sortowania. Jego średnia złożoność czasowa (czyli najszybszy przypadek posortowania) wynosi O(n*log(n)). Niestety, złożoność pesymistyczna (najgorszy przypadek dla algorytmu) nadal wynosi O(n^2). Algorytm używa metody „dziel i zwyciężaj” – rozbija rekurencyjnie tablice na podtablice.

Algorytm wygląda następująco:

1. Wybieramy dowolny element w tablicy.
2. Przenosimy na lewo elementy mniejsze od niego, a na prawo elementy, które są od niego większe.
3. Rekurencyjnie powtarzamy punkty 1. i 2. na podtablicach do momentu, aż wielkość z każdej rozpatrywanej tablicy będzie wielkości jednego elementu.

```
def czesciowe_sortowanie(tablica, lewy, prawy):
    i = (lewy - 1)
    wybrany_punkt = tablica[prawy]
    for j in range(lewy, prawy):
        if tablica[j] <= wybrany_punkt:
            i = i + 1
            tablica[i], tablica[j] = tablica[j], tablica[i]

    tablica[i + 1], tablica[prawy] = tablica[prawy], tablica[i + 1]
    return i + 1


def quick_sort(tablica, lewy, prawy):
    if len(tablica) == 1:
        return tablica
    if lewy < prawy:
        wybrany_punkt = czesciowe_sortowanie(tablica, lewy, prawy)
        quick_sort(tablica, lewy, wybrany_punkt - 1)
        quick_sort(tablica, wybrany_punkt + 1, prawy)
    return tablica

tablica = [3, 7, 1, 8, 2, 9, 6]
tabs = quick_sort(tablica.copy(), 0, len(tablica)-1)
print(tablica)
print(tabs)
```

Rys. 1.5. Kod języka programowania Python – algorytm sortowania szybkiego

```
PS D:\SOAW2> python test.py
[3, 7, 1, 8, 2, 9, 6]
[1, 2, 3, 6, 7, 8, 9]
```

Rys. 1.6. Wynik działania programu – algorytm sortowania szybkiego

Pesymistycznym przykładem tablicy dla algorytmu sortowania szybkiego (quicksort) jest tablica posortowana malejąco.

Sortowanie przez scalanie (ang. *merge sort*) to kolejny rekurencyjny algorytm stosujący metodę „dziel i zwyciężaj". Jego złożoność czasowa wynosi O(n*log(n)).

Algorytm wygląda następująco:

1. Tablicę dzielimy na dwie równe części (jeżeli jest nieparzysta liczba elementów, to jedna część będzie miała o jeden element mniej).
2. Powtarzamy to tak długo, aż otrzymamy oddzielne elementy.
3. Łączymy najbliższe pary podtablic, sortując je.
4. Powtarzamy krok 3. tak długo, aż zostanie jedna tablica.

```
def merge_sort(tablica):
    if len(tablica) > 1:
        pivot = len(tablica)//2
        lewa = tablica[pivot:]
        prawa = tablica[:pivot]
        merge_sort(lewa)
        merge_sort(prawa)
        # i - aktualny indeks, l - lewy indeks, p - prawy indeks
        i = l = p = 0
        while l < len(lewa) and p < len(prawa):
            if lewa[l] <prawa[p]:
                tablica[i] = lewa[l]
                l+=1
            else:
                tablica[i] = prawa[p]
                p+=1
            i+=1

        for z in range(l, len(lewa)):
            tablica[i] = lewa[l]
```

Rys. 1.7. Kod języka programowania Python – algorytm sortowania przez scalanie (część 1)

```
        for z in range(l, len(lewa)):
            tablica[i] = lewa[l]
            i += 1
            l += 1

        for z in range(p, len(prawa)):
            tablica[i] = prawa[p]
            i += 1
            p += 1

    return tablica


tablica = [3, 7, 1, 8, 2, 9, 6]
tabs = merge_sort(tablica.copy())
print(tablica)
print(tabs)
```

Rys. 1.8. Kod języka programowania Python – algorytm sortowania przez scalanie (część 2)

```
PS D:\SOAW2> python test.py
[3, 7, 1, 8, 2, 9, 6]
[1, 2, 3, 6, 7, 8, 9]
```

Rys. 1.9. Wynik działania programu – algorytm sortowania przez scalanie

1.2. Algorytmy wyszukiwania

W programowaniu mamy kilka metod na przeszukiwanie zbiorów w celu znalezienia interesującego nas elementu.

Najczęstsze algorytmy przeszukiwania to:

- przeszukiwanie liniowe,
- przeszukiwanie na drzewie binarnym,
- przeszukiwanie binarne.

Przeszukiwanie liniowe

Ma ono złożoność czasową O(n). Przeszukuje cały zbiór element po elemencie, dopóki nie znajdzie szukanego elementu.

```
def przeszukiwanie_liniowe(tablica, szukana):
    index = -1
    for i in range(0, len(tablica)):
        if tablica[i] == szukana:
            index = i
            break

    return index


tablica = [3, 7, 1, 8, 2, 9, 6]
index = przeszukiwanie_liniowe(tablica, 8)
print(tablica)
print("Index szukanego elementu: "+str(index))
```

Rys. 1.10. Kod języka programowania Python – algorytm przeszukiwania liniowego

```
PS D:\SOAW2> python test.py
[3, 7, 1, 8, 2, 9, 6]
Index szukanego elementu: 3
```

Rys. 1.11. Wynik działania programu – algorytm przeszukiwania liniowego

Przeszukiwanie na drzewie binarnym

Drzewa binarne mają taką własność, że w lewym poddrzewie są wartości mniejsze, a w prawym poddrzewie – wartości większe. Używając tej własności, możemy łatwo wywnioskować, że złożoność czasowa tego algorytmu to nie więcej niż wysokość drzewa binarnego – liczby połączeń między korzeniem a najdalej wysuniętym od niego liściem.

Przeszukiwanie binarne

Wymaga od nas posortowanego zbioru. Algorytm ma złożoność czasową O(log(n)). Wygląda on następująco:

1. Dzieli uporządkowany zbiór na pół. Sprawdza, w której połowie mieści się szukany element (czy jest większy / mniejszy od elementu środkowego).
2. Dzieli wybraną połowę po raz kolejny na pół i powtarza to działanie tak długo, aż znajdzie szukany element.

```
def przeszukiwanie_binarne(posortowana_tablica,
lewa, prawa,  szukana):
    if lewa <= prawa:
        srodek = (prawa+lewa)//2
        if posortowana_tablica[srodek] ==
        szukana:
            return srodek
        if posortowana_tablica[srodek] >
        szukana:
            return przeszukiwanie_binarne
            (posortowana_tablica, lewa,
            srodek-1, szukana)
        else:
            return przeszukiwanie_binarne
            (posortowana_tablica, srodek+1,
            prawa, szukana)
    else:
        return -1


tablica = [1, 2, 3, 6, 7, 8, 9]
index = przeszukiwanie_binarne(tablica, 0, len
(tablica), 8)
print(tablica)
print("Index szukanego elementu: "+str(index))
```

Rys. 1.12. Kod języka programowania Python – algorytm przeszukiwania binarnego

```
PS D:\SOAW2> python test.py
[1, 2, 3, 6, 7, 8, 9]
Index szukanego elementu: 5
```

Rys. 1.13. Wynik działania programu – algorytm przeszukiwania binarnego

SPRAWDŹ SWOJĄ WIEDZĘ

1. Opisz algorytm sortowania bąbelkowego.
2. Scharakteryzuj algorytm sortowania szybkiego.
3. Omów działanie algorytmu przeszukiwania liniowego.
4. Opisz działanie algorytmu sortowania przez wstawianie.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1.
   - W języku programowania Python utwórz tablicę o wielkości 10 000 elementów.
   - Wypełnij utworzoną tablicę losowymi liczbami całkowitymi z zakresu od 1 do 1000. Warto użyć funkcji **`randint`** z biblioteki wbudowanej **`random`**. Zastosowanie wymienionej funkcji może wyglądać następująco:

     ```
     random.randint(dolny_zakres, górny_zakres)
     ```

   - Posortuj tablicę z użyciem:
     A. sortowania bąbelkowego,
     B. algorytmu quicksort.
   - Zmierz czasy wykonania obu algorytmów za pomocą biblioteki time.

   Aby zmierzyć czas, musisz użyć następującej konstrukcji w kodzie:

   ```
   czas_poczatkowy = time.time()
   instrukcje….
   czas_koncowy = time.time() - czas_poczatkowy
   ```

2. Zmień kod algorytmu sortowania bąbelkowego z rys. 1.3 w taki sposób, aby sortował elementy w tablicy w kolejności malejącej.

2 Wzorce projektowe

ZAGADNIENIA

- Wzorce projektowe
- Fasada
- Metoda szablonowa

2.1. Wzorce projektowe

Tworząc oprogramowanie, potrzebujemy różnych typów rozwiązań dla zadanych problemów. Abyśmy nie musieli tworzyć ich od nowa, wymyślono wzorce projektowe, na których bazie możemy opracowywać aplikacje.

Wzorzec projektowy to sprawdzone w praktyce rozwiązanie problemu projektowania oprogramowania. Zazwyczaj jest to szkic klas i powiązań między nimi tak skonstruowany, aby ułatwić tworzenie, modyfikację i utrzymywanie aplikacji oraz zachowanie przejrzystości kodu źródłowego, nawet w razie zmieniających się programistów (osób) w projekcie.

W świecie programowania istnieje wiele wzorców projektowych. Jednymi z częściej używanych są:

- fasada,
- metoda szablonowa,
- kompozyt,
- fabryka,
- budowniczy,
- obserwator,
- adapter,
- mediator.

2.2. Fasada

Jest to wzorzec projektowy, który zamienia skomplikowaną strukturę klas w prosty interfejs użytkownika, oferujący użytkownikowi wyłącznie potrzebne funkcjonalności.

Fasada to prosty i ładnie wyglądający interfejs, pod którym wykonują się skomplikowane procesy.

Aby łatwo zobrazować problem fasady, załóżmy, że jest nim robienie śniadania przez inteligentny dom. Użytkownika nie interesuje, jak przebiega proces. Chce po prostu jak najszybciej zasiąść do śniadania. Dlatego dostarczamy mu jedną prostą metodę, po której wywołaniu program zrobi pełne śniadanie.

```
class FasadaPrzygotujSniadanie:
    def __init__(self, operacja_eksperes,
    operacja_toster):
        self.operacja_eksperes = operacja_eksperes
        self.operacja_toster = operacja_toster

    def rob_sniadanie(self):
        print("Ładowanie podsystemów")
        self.operacja_eksperes.wlacz_ekspress()
        self.operacja_toster.wlacz_toster()
        print("Zaczynam działać")
        self.operacja_eksperes.rob_kawe()
        self.operacja_toster.rob_tosty()
```

Rys. 2.1. Kod języka programowania Python – przykład fasady (część 1)

```
class OperacjaEkspres:
    def wlacz_ekspress(self):
        print("Ekspres włączony.\nEkspres gotowy do
        działania")

    def rob_kawe(self):
        print("Kawa gotowa!")


class OperacjaToster:
    def wlacz_toster(self):
        print("Toster włączony.\nToster gotowy do
        działania")

    def rob_tosty(self):
        print("Tosty gotowe!")
```

Rys. 2.2. Kod języka programowania Python – przykład fasady (część 2)

```
if __name__ == "__main__":
    ekspres = OperacjaEkspres()
    toster = OperacjaToster()
    fasada = FasadaPrzygotujSniadanie(ekspres, toster)
    fasada.rob_sniadanie()
```

Rys. 2.3. Kod języka programowania Python – przykład fasady (część 3)

```
PS D:\SOAW2> python fasada.py
Ładowanie podsystemów
Ekspres włączony.
Ekspres gotowy do działania
Toster włączony.
Toster gotowy do działania
Zaczynam działać
Kawa gotowa!
Tosty gotowe!
```

Rys. 2.4. Wynik działania programu – przykład fasady w języku programowania Python

Jak widzimy na powyższych przykładach, użytkownik ma tylko jedną możliwość z klasy „FasadaPrzygotujSniadanie” – czyli **rob_sniadanie**.

2.3. Metoda szablonowa

Ten wzorzec projektowy jest jednym z najczęściej używanych szablonów. Wiele frameworków Pythona z niego korzysta, m.in.: matplotlib, kivy i tkinter. Taki wzorzec to zazwyczaj klasa abstrakcyjna wymuszająca na programiście nadpisanie funkcjonalności klasy bazowej w klasach pochodnych.

Na bazie takiego wzorca projektowego może istnieć fabryka składająca różne modele samochodów z gotowej listy procedur lub aplikacja tworząca odpowiednie strony internetowe według gotowych schematów.

```python
# w pythonie, aby napisać metody abstrakcyjne, musimy
zaimportować bibliotekę abc
from abc import abstractmethod

class AbstrakcyjnySzablon:
    def metoda_szablonowa(self):
        self.procedura1()
        self.procedura2()
        self.procedura3()
        self.procedura4()
        self.procedura5()
        self.procedura6()

    def procedura1(self):
        print("Wykonuję procedurę nr 1")

    @abstractmethod
    def procedura2(self):
        pass

    # punkt procedury, który możemy czasem pominąć
    def procedura3(self):
        pass
```

Rys. 2.5. Kod języka programowania Python – przykład metody szablonowej (część 1)

CIEKAWOSTKA

Przypomnienie!
Aby klasa była abstrakcyjna w języku programowania Python, trzeba:

- zaimportować z biblioteki „abc" moduł „abstractmethod";
- użyć flagi własności „@abstractmethod" przed metodą, z której zamierza się utworzyć metodę wirtualną – nada klasie abstrakcyjność.

Pamiętaj: nie można utworzyć obiektu z klasy abstrakcyjnej.

```
    def procedura4(self):
        print("Wykonuję procedurę nr 4")

    @abstractmethod
    def procedura5(self):
        pass

    def procedura6(self):
        print("Wykonuję procedurę nr 6")


# Zaznaczamy w nawiasie dziedziczenie klasy po klasie
AbstrakcyjnySzablon
class Model1(AbstrakcyjnySzablon):
    def procedura2(self):
        print("Procedura nr 2 wywołana przez Model1")

    def procedura5(self):
        print("Procedura nr 5 wywołana przez Model1")
```

Rys. 2.6. Kod języka programowania Python – przykład metody szablonowej (część 2)

CIEKAWOSTKA

Aby nowa klasa „Model1", dziedzicząc po klasie „AbstrakcyjnySzablon", przestała być klasą abstrakcyjną, trzeba w niej nadpisać wszystkie abstrakcyjne metody rodzica, nazwane kolejno: „procedura2" i „procedura5".
Tak samo będzie w przypadku klasy „Model2". Należy nadpisać metody wirtualne rodzica: „procedura2", „procedura5".
Trzeba pamiętać, że w klasie dziedziczącej można dodać kolejne metody – nieodziedziczone po rodzicu. W klasie „Model2" jest to metoda „procedura3".
W Pythonie wystarczy nadpisać klasy rodzica bez podawania odpowiednich flag własności (ang. *property*), np. @override bądź @abstractmethod. Co ważne, jest to wymagane w wielu innych językach programowania obiektowego (np. Java).

```python
class Model2(AbstrakcyjnySzablon):
    def procedura2(self):
        print("Procedura nr 2 wywołana przez Model2")

    def procedura5(self):
        print("Procedura nr 5 wywołana przez Model2")

    def procedura3(self):
        print("Procedura dodatkowa nr 3 wywołana przez Model2")


if __name__ == "__main__":
    Model1().metoda_szablonowa()
    print()
    Model2().metoda_szablonowa()
```

Rys. 2.7. Kod języka programowania Python – przykład metody szablonowej (część 3)

```
PS D:\SOAW2> python metoda_szablonowa.py
Wykonuję procedurę nr 1
Procedura nr 2 wywołana przez Model1
Wykonuję procedurę nr 4
Procedura nr 5 wywołana przez Model1
Wykonuję procedurę nr 6

Wykonuję procedurę nr 1
Procedura nr 2 wywołana przez Model2
Procedura dodatkowa nr 3 wywołana przez Model2
Wykonuję procedurę nr 4
Procedura nr 5 wywołana przez Model2
Wykonuję procedurę nr 6
```

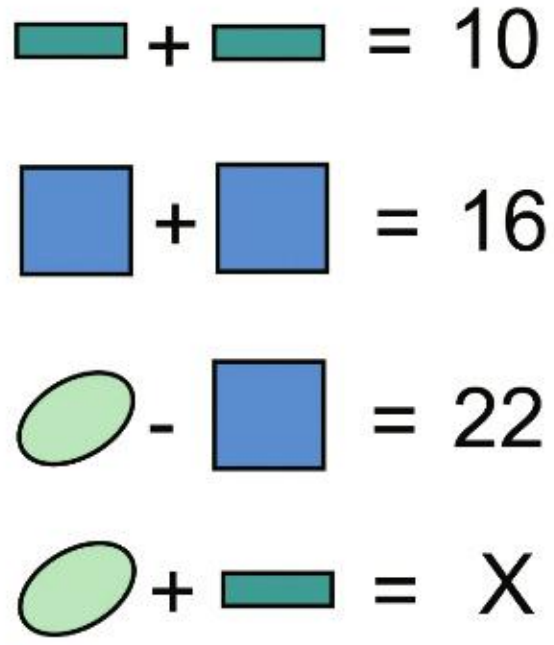
Rys. 2.8. Wynik działania programu – przykład metody szablonowej w języku programowania Python

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień cztery nazwy wzorców projektowych.
2. Do czego można użyć wzorca projektowego „fasada”?
3. Opisz na przykładzie możliwe zastosowanie wzorca projektowego „metoda szablonowa”.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Przyjrzyj się rys. 2.9. Dokładnie przeanalizuj poszczególne linijki w poszukiwaniu wzorców. Na bazie zawartych na nim wzorców wylicz zmienną X. Odpowiedź uzasadnij.



Rys. 2.9. Wynik działania programu – przykład metody szablonowej w języku

2. Na bazie rys. 2.9 przygotuj co najmniej pięć podobnych grafik (najlepiej w edytorze tekstu lub programie graficznym Inkscape bądź GIMP). W każdym przykładzie wykorzystaj co najmniej trzy spośród czterech podstawowych działań matematycznych:
   - dodawanie,
   - odejmowanie,
   - dzielenie,
   - mnożenie.

3 Prawo autorskie w dziedzinie programowania

ZAGADNIENIA

- Autorskie prawa osobiste i majątkowe
- Czas trwania praw autorskich
- Konsekwencje naruszenia prawa autorskiego
- Elementy własności intelektualnej (dobra niematerialne, własności przemysłowe)
- Typy licencji oprogramowania

3.1. Autorskie prawa osobiste i majątkowe

W polskim prawie autorskim można wyodrębnić:

- autorskie prawa osobiste,
- autorskie prawa majątkowe.

Autorskie prawa osobiste

Uregulowano je w *Ustawie z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych* (t.j., DzU z 2019 r., poz. 1231)[1].

Art. 1.

1. Przedmiotem prawa autorskiego jest każdy przejaw działalności twórczej o indywidualnym charakterze, ustalony w jakiejkolwiek postaci, niezależnie od wartości, przeznaczenia i sposobu wyrażenia (utwór).
2. W szczególności przedmiotem prawa autorskiego są utwory:
   1) wyrażone słowem, symbolami matematycznymi, znakami graficznymi (literackie, publicystyczne, naukowe, kartograficzne oraz programy komputerowe);
   2) plastyczne;
   3) fotograficzne;
   4) lutnicze;
   5) wzornictwa przemysłowego;
   6) architektoniczne, architektoniczno-urbanistyczne i urbanistyczne;
   7) muzyczne i słowno-muzyczne;
   8) sceniczne, sceniczno-muzyczne, choreograficzne i pantomimiczne;
   9) audiowizualne (w tym filmowe).

[1] Na stronie http://www.prawoautorskie.gov.pl/ możesz znaleźć oficjalne rządowe opracowania dotyczące omawianych zagadnień.

2[1]. Ochroną objęty może być wyłącznie sposób wyrażenia; nie są objęte ochroną odkrycia, idee, procedury, metody i zasady działania oraz koncepcje matematyczne.
3. Utwór jest przedmiotem prawa autorskiego od chwili ustalenia, chociażby miał postać nieukończoną.
4. Ochrona przysługuje twórcy niezależnie od spełnienia jakichkolwiek formalności.

W rozdziale 3. ustawy o prawie autorskim i prawach pokrewnych uregulowano kwestie autorskich praw osobistych. Celem autorskich praw osobistych jest ochrona uprawnień związanych z dziełem innych niż ekonomiczne. Co najważniejsze, więź twórcy z utworem ma trwałe, nieograniczone i niezbywalne powiązanie. Próby ograniczenia autorskich praw osobistych wbrew woli autora lub działania zmierzające do pozbawienia go tych praw są bezskuteczne i nieważne.

Oddział 1.
Autorskie prawa osobiste

Art. 16.

Jeżeli ustawa nie stanowi inaczej, autorskie prawa osobiste chronią nieograniczoną w czasie i niepodlegającą zrzeczeniu się lub zbyciu więź twórcy z utworem, a w szczególności prawo do:

1) autorstwa utworu;
2) oznaczenia utworu swoim nazwiskiem lub pseudonimem albo do udostępniania go anonimowo;
3) nienaruszalności treści i formy utworu oraz jego rzetelnego wykorzystania;
4) decydowania o pierwszym udostępnieniu utworu publiczności;
5) nadzoru nad sposobem korzystania z utworu.

Oddział 2.
Autorskie prawa majątkowe

Art. 17.

Jeżeli ustawa nie stanowi inaczej, twórcy przysługuje wyłączne prawo do korzystania z utworu i rozporządzania nim na wszystkich polach eksploatacji oraz do wynagrodzenia za korzystanie z utworu.

Autorskie prawa majątkowe

W przeciwieństwie do praw osobistych autorskie prawa majątkowe są zbywalne, co wynika z zawartego w rozdziale 5. artykułu 41. wspomnianej ustawy.

Przejście autorskich praw majątkowych

Art. 41.

1. Jeżeli ustawa nie stanowi inaczej:
   1) autorskie prawa majątkowe mogą przejść na inne osoby w drodze dziedziczenia lub na podstawie umowy;
   2) nabywca autorskich praw majątkowych może przenieść je na inne osoby, chyba że umowa stanowi inaczej.
2. Umowa o przeniesienie autorskich praw majątkowych lub umowa o korzystanie z utworu, zwana dalej „licencją", obejmuje pola eksploatacji wyraźnie w niej wymienione.
3. Nieważna jest umowa w części dotyczącej wszystkich utworów lub wszystkich utworów określonego rodzaju tego samego twórcy mających powstać w przyszłości.
4. Umowa może dotyczyć tylko pól eksploatacji, które są znane w chwili jej zawarcia.
5. Twórca utworu wykorzystanego lub włączonego do utworu audiowizualnego oraz utworu wchodzącego w skład utworu zbiorowego, po powstaniu nowych sposobów eksploatacji utworów, nie może bez ważnego powodu odmówić udzielenia zezwolenia na korzystanie z tego utworu w ramach utworu audiowizualnego lub utworu zbiorowego na polach eksploatacji nieznanych w chwili zawarcia umowy.

Art. 42.

Jeżeli autorskie prawa majątkowe jednego ze współtwórców miałyby przypaść Skarbowi Państwa jako spadkobiercy ustawowemu, część ta przechodzi na pozostałych przy życiu współtwórców lub ich następców prawnych, stosownie do wielkości ich udziałów.

Art. 43.

1. Jeżeli z umowy nie wynika, że przeniesienie autorskich praw majątkowych lub udzielenie licencji nastąpiło nieodpłatnie, twórcy przysługuje prawo do wynagrodzenia.
2. Jeżeli w umowie nie określono wysokości wynagrodzenia autorskiego, wysokość wynagrodzenia określa się z uwzględnieniem zakresu udzielonego prawa oraz korzyści wynikających z korzystania z utworu.

Art. 44.

W razie rażącej dysproporcji między wynagrodzeniem twórcy a korzyściami nabywcy autorskich praw majątkowych lub licencjobiorcy twórca może żądać stosownego podwyższenia wynagrodzenia przez sąd.

Art. 45.

Jeżeli umowa nie stanowi inaczej, twórcy przysługuje odrębne wynagrodzenie za korzystanie z utworu na każdym odrębnym polu eksploatacji.

3.2. Czas trwania praw autorskich

Autorskie prawa majątkowe są dość ściśle ograniczone w czasie. Ustalono go w rozdziale 4. wspomnianej ustawy.

Czas trwania autorskich praw majątkowych

Art. 36.

Z zastrzeżeniem wyjątków przewidzianych w ustawie, autorskie prawa majątkowe gasną z upływem lat siedemdziesięciu:

1) od śmierci twórcy, a do utworów współautorskich – od śmierci współtwórcy, który przeżył pozostałych;
2) w odniesieniu do utworu, którego twórca nie jest znany – od daty pierwszego rozpowszechnienia, chyba że pseudonim nie pozostawia wątpliwości co do tożsamości autora lub jeżeli autor ujawnił swoją tożsamość;
3) w odniesieniu do utworu, do którego autorskie prawa majątkowe przysługują z mocy ustawy innej osobie niż twórca – od daty rozpowszechnienia utworu, a gdy utwór nie został rozpowszechniony – od daty jego ustalenia;
4) w odniesieniu do utworu audiowizualnego – od śmierci najpóźniej zmarłej z wymienionych osób: głównego reżysera, autora scenariusza, autora dialogów, kompozytora muzyki skomponowanej do utworu audiowizualnego.
5) w odniesieniu do utworu słowno-muzycznego, jeżeli utwór słowny i utwór muzyczny zostały stworzone specjalnie dla danego utworu słowno-muzycznego – od śmierci później zmarłej z wymienionych osób: autora utworu słownego albo kompozytora utworu muzycznego.

Art. 37.

Jeżeli bieg terminu wygaśnięcia autorskich praw majątkowych rozpoczyna się od rozpowszechnienia utworu, a utwór rozpowszechniono w częściach, odcinkach, fragmentach lub wkładkach, bieg terminu liczy się oddzielnie od daty rozpowszechnienia każdej z wymienionych części.

Art. 39.

Czas trwania autorskich praw majątkowych liczy się w latach pełnych następujących po roku, w którym nastąpiło zdarzenie, od którego zaczyna się bieg terminów określonych w art. 36 i art. 37.

3.3. Konsekwencje naruszenia prawa autorskiego

Naruszenie autorskich praw osobistych opisano w rozdziale 8. omawianej ustawy.

Ochrona praw osobistych

Art. 78.

1. Twórca, którego autorskie prawa osobiste zostały zagrożone cudzym działaniem, może żądać zaniechania tego działania. W razie dokonanego naruszenia może także żądać, aby osoba, która dopuściła się naruszenia, dopełniła czynności potrzebnych do usunięcia jego skutków, w szczególności aby złożyła publiczne oświadczenie o odpowiedniej treści i formie. Jeżeli naruszenie było zawinione, sąd może przyznać twórcy odpowiednią sumę pieniężną tytułem zadośćuczynienia za doznaną krzywdę lub – na żądanie twórcy – zobowiązać sprawcę, aby uiścił odpowiednią sumę pieniężną na wskazany przez twórcę cel społeczny.
2. Jeżeli twórca nie wyraził innej woli, po jego śmierci z powództwem o ochronę autorskich praw osobistych zmarłego może wystąpić małżonek, a w jego braku kolejno: zstępni, rodzice, rodzeństwo, zstępni rodzeństwa.
3. Jeżeli twórca nie wyraził innej woli, osoby wymienione w ust. 2 są uprawnione w tej samej kolejności do wykonywania autorskich praw osobistych zmarłego twórcy.
4. Jeżeli twórca nie wyraził innej woli, z powództwem, o którym mowa w ust. 2, może również wystąpić stowarzyszenie twórców właściwe ze względu na rodzaj twórczości lub organizacja zbiorowego zarządzania prawami autorskimi lub prawami pokrewnymi, która zarządzała prawami autorskimi zmarłego twórcy.

Naruszenie autorskich praw majątkowych opisano w rozdziale 9. omawianej ustawy.

Ochrona autorskich praw majątkowych

Art. 79.

1. Uprawniony, którego autorskie prawa majątkowe zostały naruszone, może żądać od osoby, która naruszyła te prawa:
   1) zaniechania naruszania;
   2) usunięcia skutków naruszenia;
   3) naprawienia wyrządzonej szkody:
      a) na zasadach ogólnych albo
      b) poprzez zapłatę sumy pieniężnej w wysokości odpowiadającej dwukrotności, a w przypadku gdy naruszenie jest zawinione – trzykrotności stosownego wynagrodzenia, które w chwili jego dochodzenia byłoby należne tytułem udzielenia przez uprawnionego zgody na korzystanie z utworu;
   4) wydania uzyskanych korzyści.

2. Niezależnie od roszczeń, określonych w ust. 1, uprawniony może się domagać jednokrotnego albo wielokrotnego ogłoszenia w prasie oświadczenia o odpowiedniej treści i formie lub podania do publicznej wiadomości części albo całości orzeczenia sądu wydanego w rozpatrywanej sprawie, w sposób i w zakresie określonym przez sąd.
3. Sąd może nakazać osobie, która naruszyła autorskie prawa majątkowe, na jej wniosek i za zgodą uprawnionego, w przypadku gdy naruszenie jest niezawinione, zapłatę stosownej sumy pieniężnej na rzecz uprawnionego, jeżeli zaniechanie naruszania lub usunięcie skutków naruszenia byłoby dla osoby naruszającej niewspółmiernie dotkliwe.
4. Sąd, rozstrzygając o naruszeniu prawa, może orzec na wniosek uprawnionego o bezprawnie wytworzonych przedmiotach oraz środkach i materiałach użytych do ich wytworzenia, w szczególności może orzec o ich wycofaniu z obrotu, przyznaniu uprawnionemu na poczet należnego odszkodowania lub zniszczeniu. Orzekając, sąd uwzględnia wagę naruszenia oraz interesy osób trzecich.
5. Domniemywa się, że środki i materiały, o których mowa w ust. 4, są własnością osoby, która naruszyła autorskie prawa majątkowe.
6. Przepis ust. 1 stosuje się odpowiednio w przypadku usuwania lub obchodzenia technicznych zabezpieczeń przed dostępem, zwielokrotnianiem lub rozpowszechnianiem utworu, jeżeli działania te mają na celu bezprawne korzystanie z utworu.
7. Przepisy ust. 1 i 2 stosuje się odpowiednio w przypadku usuwania lub zmiany bez upoważnienia jakichkolwiek elektronicznych informacji na temat zarządzania prawami autorskimi lub prawami pokrewnymi, a także świadomego rozpowszechniania utworów z bezprawnie usuniętymi lub zmodyfikowanymi takimi informacjami.

3.4. Elementy własności intelektualnej (dobra niematerialne, własności przemysłowe)

Prawa na dobrach niematerialnych to wyłączne prawo do wytworów ludzkiego umysłu, czyli właściwie do wszystkiego, co potrafimy stworzyć, a co jest unikatowe czy też jeszcze nieistniejące. Zauważ, że jeżeli wymyślisz coś, co już istnieje, to oczywiście masz do tego prawo – własne, ale już niekoniecznie w ujęciu publicznym. Jeżeli ktoś inny zastrzegł daną rzecz, to on będzie miał do tego wyłączne prawo. Prawa na dobrach niematerialnych to obszar obejmujący różnego rodzaju twórczość, m.in. w zakresie stworzenia:

- dzieł literackich, malarskich, muzycznych,
- wzorów użytkowych,
- wzorów przemysłowych,
- różnego rodzaju wynalazków,
- całej firmy – prawo do firmy czy też do znaku towarowego (nazwy, marki itp.),
- określonego (unikatowego) produktu.

W skład omawianego zagadnienia wchodzą dobra niematerialne oparte na:
- prawie własności przemysłowej,
- prawie autorskim i prawach pokrewnych.

W programowaniu najważniejsze jest to, że w skład tego obszaru wchodzą także wykorzystywane na co dzień technologie informatyczne (komputerowe, internetowe, serwerowe itd.), w szczególności związane z:
- używanym sprzętem IT i wykorzystywanymi w nim technologiami,
- oprogramowaniem komputerowym – systemy operacyjne, programy użytkowe itp.,
- technologiami tworzenia oprogramowania (komputerowego, webowego, na urządzenia przenośne itp.), czyli prawami do języków programowania i środowisk programistycznych związanych z wytwarzaniem oprogramowania.

Prawo własności intelektualnej opiera się na wielu aktach prawnych krajowych, ale także bazujących na ogólnym prawie Unii Europejskiej. Są to m.in.:
- *Ustawa z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych*,
- *Ustawa z dnia 30 czerwca 2000 r. Prawo własności przemysłowej* (t.j., DzU z 2021 r., poz. 324),
- *Traktat Światowej Organizacji Własności Intelektualnej o Prawie Autorskim, sporządzony w Genewie dnia 20 grudnia 1996 r.* (DzU z 2005 r. nr 3, poz. 12),
- *Obwieszczenie Marszałka Sejmu Rzeczypospolitej Polskiej z dnia 11 września 2019 r. w sprawie ogłoszenia jednolitego tekstu ustawy o rzecznikach patentowych* (t.j., DzU z 2020 r., poz. 288).

Instytucją, która w Polsce nadzoruje i certyfikuje własność intelektualną, jest Urząd Patentowy RP (http://www.uprp.pl). Na portalu tej organizacji można znaleźć wiele przydatnych informacji z zakresu praw własności intelektualnej. Są tam m.in. serie bezpłatnych książek elektronicznych do pobrania ze strony https://uprp.gov.pl/pl/publikacje/publikacje-bezplatne. Warto tam zajrzeć i zapoznać się z najważniejszymi zagadnieniami.

Istnieje jeszcze co najmniej jeden ciekawy serwis zajmujący się obszarem wiedzy opisywanym w tej części podręcznika – Espacenet. Wyszukiwanie patentów, Serwis prowadzony we współpracy z EPO (ang. *European Patent Office*) – https://pl.espacenet.com/. Jego pierwotna strona Espacenet patent search znajduje się pod adresem https://www.epo.org/searching-for-patents/technical/espacenet.html. Obejmuje ona obszar całej Unii Europejskiej, umożliwia dostęp do informacji o patentach zarejestrowanych na terenie UE i reszty świata.

3.5. Typy licencji oprogramowania

Licencja na oprogramowanie to dokument (umowa) określający zasady użytkowania danego programu komputerowego (aplikacji) na określonych w niej polach eksploatacji. **Polami eksploatacji** mogą być m.in.:

- publikacja w internecie pod wskazanym adresem WWW (domeną),
- utrwalanie i zwielokrotnianie egzemplarzy książki techniką drukarską i wprowadzanie do obrotu handlowego egzemplarzy drukowanych,
- zapis dźwiękowy i dystrybucja plików elektronicznych, zawierających książkę w wersji dźwiękowej,
- publiczne udostępnianie przez odczyt do publiczności zgromadzonej w sali konferencyjnej,
- używanie pojedynczej kopii (np. programu komputerowego) na jednym stanowisku roboczym.

Z punktu widzenia odpłatności można podzielić licencje na:

- **płatne** – zawsze gdy chcesz używać oprogramowania o tym typie licencjonowania, musisz uiścić określoną kwotę, będącą wynagrodzeniem za korzystanie z efektów cudzej pracy;
- **częściowo płatne** – oprogramowanie licencjonowane według zasad tego punktu jest bezpłatne tylko po spełnieniu dodatkowego warunku – np. tylko do użytku prywatnego (w ten sposób jest dystrybuowanych m.in. kilka programów antywirusowych), za to przy używaniu komercyjnym – na firmę trzeba wykupić (opłacić) prawo do legalnego użytkowania;
- **bezpłatne** – zwykle zupełnie bezpłatnie, bez dodatkowych warunków można używać tego oprogramowania, ale czasem uiszcza się pewne klauzule niematerialne – jak np. uznanie autorstwa, czyli poinformowanie podczas dystrybucji własnego oprogramowania o twórcach programów (modułów), które wykorzystano do stworzenia aplikacji.

O tym, jakim typem licencji jest objęty dany program komputerowy, decyduje twórca, zwykle przez dołączenie do dostarczanej aplikacji (wraz z kodami programistycznymi) stosownych zapisów w dokumencie „Licencja". Zadaniem użytkownika jest zapoznanie się z warunkami licencji, zrozumienie ich oraz zaakceptowanie i stosowanie się do stosownych zapisów licencyjnych. Pamiętaj, że: **przez używanie programu komputerowego oświadczasz, że zapoznałeś się z warunkami licencyjnymi i je akceptujesz**.

Jeżeli ktoś przyłapałby cię na nielegalnym używaniu programu, nie możesz się tłumaczyć, że nie zapoznałeś się z warunkami licencyjnymi lub ich nie zrozumiałeś.



Rys. 3.1. Licencja – zasady korzystania z programów komputerowych

Korzystanie z oprogramowania komercyjnego (odpłatnego) w wielu przypadkach ma oczywiście pełne uzasadnienie. Skoro ktoś wykonał jakąś pracę, to może chcieć otrzymać za nią wynagrodzenie. Tematu płatnych licencji nie będziemy poruszali. Różnią się one między sobą dość znacznie i za każdym razem trzeba po prostu dokładnie zapoznać się z warunkami takiego sposobu udostępniania. Podobnie wygląda sytuacja z programami, które na bazie licencji są bezpłatne (z zachowaniem określonych warunków). Trzeba zawsze doczytać, jakie konkretnie są postanowienia licencyjne.

Na rynku istnieją określone zasady licencjonowania dotyczące niektórych bezpłatnych typów udostępniania wartości niematerialnych i prawnych. Gdy widzisz, że twórca umieścił przy swoim oprogramowaniu określony typ licencji, od razu wiesz, czego nie możesz robić, a co jest dozwolone. Poniżej znajdziesz krótki opis kilku najpopularniejszych definicji i opisów licencji ze świata wolnego oprogramowania i bezpłatnego oprogramowania. Co ważne, te dwa pojęcia zdecydowanie nie są tożsame.

Bezpłatność oprogramowania dotyczy możliwości korzystania z programu bez konieczności wydawania na niego pieniędzy.

Wolne oprogramowanie jest rozumiane zdecydowanie szerzej, gdyż wolność rozpatrywana jest na kilku poziomach (poniższy tekst opracowano na podstawie *The Free Software Foundation – FSF*, https://www.gnu.org/philosophy/free-sw.html):

„Program jest wolnym oprogramowaniem, jeśli użytkownicy programu mają cztery niezbędne wolności:

- Wolność do uruchamiania programu w dowolnym celu (wolność 0).
- Wolność do analizowania, jak działa program, i zmieniania go, aby użytkownik robił, co i jak chce. Warunkiem koniecznym jest dostęp do kodu źródłowego (wolność 1).
- Wolność do rozpowszechniania kopii, aby pomagać innym (wolność 2).
- Wolność do udoskonalania programu i publicznego rozpowszechniania własnych ulepszeń, dzięki czemu może z nich skorzystać cała społeczność (wolność 3)".

Warto podkreślić, że obok pojęcia „wolnego oprogramowania" mamy na rynku informatycznym także „oprogramowanie otwarte". Otwarta licencja skupia się na dwóch ostatnich wolnościach, a wolna – na dwóch pierwszych. **Otwarta licencja** pozwala na dalszą dystrybucję powstałego dzieła na dowolnej licencji, podczas gdy wolna nie umożliwia udostępniania dzieła na innej licencji niż wolna.

Wolne oprogramowanie nie zawsze oznacza, że jest ono bezpłatne, bo wolność nie jest rozpatrywana w kategoriach ceny. Niska cena lub jej brak wcale nie oznacza większej wolności. O tym decydują cztery wcześniej wymienione punkty. O rozwój idei wolnego oprogramowania dba fundacja The Free Software Foundation (FSF, https://www.fsf.org/). Pod jej pieczą jest wiele różnorodnych technologii, a chyba najbardziej znaną jest Gnu System operacyjny, czyli dokładniej dystrybucje GNU/Linuksa. Jeden z rodzajów licencji jest bezpośrednio związany z FSF – Powszechna Licencja Publiczna GNU (GNU GPL). Szczegółowe informacje na jej temat znajdziesz na stronie projektu https://www.gnu.org/licenses/license-list.html#SoftwareLicenses. Na wskazanej stronie zamieszczono podział istniejących licencji na zupełnie wolne i pozostałe, które w mniejszym lub większym stopniu mają ograniczenia prawne. Wśród licencji wolnych i równocześnie bezpłatnych najpopularniejsze są:

- Powszechna Licencja Publiczna GNU (GPL – General Public License) – jej najnowsza wersja to GNUGPLv3 (https://opensource.org/licenses/LGPL-3.0);
- Boost Software License – tzw. boost (https://opensource.org/licenses/BSL-1.0);
- Licencja FreeBSD (https://directory.fsf.org/wiki/License:BSD-2-Clause-FreeBSD);

- Licencja MIT (X11) – jedna z najbardziej liberalnych licencji (https://opensource.org/licenses/mit-license.php);
- Apache License (https://opensource.org/licenses/Apache-2.0);
- BSD-3-Clause (https://opensource.org/licenses/BSD-3-Clause);
- Licencje Creative Commons (CC) – strona organizacji związanej z tym sposobem udostępniania utworów (https://creativecommons.org/).

Wykaz dostępnych licencji oprogramowania możesz znaleźć na stronie Open Source Initiative (OSI) – https://opensource.org/licenses/alphabetical.

Do tematu licencji na używane oprogramowanie będziemy jeszcze powracać w tym podręczniku, szczególnie często w czwartym rozdziale, w którym podczas przygotowania paczki na system Android na urządzenia mobilne trzeba będzie pobrać i zainstalować wiele oprogramowania. Rozwiązań możliwych do zastosowania podczas tego procesu jest sporo. Właśnie wtedy trzeba uważać, aby nieopatrznie nie użyć rozwiązania, które jest związane z odpłatnością lub nawet z niemożliwością prawną jego wykorzystania. W trzecim rozdziale pojawi się podobna sytuacja, także trzeba będzie uważać na prawa firm trzecich do ich autorskich rozwiązań.

Zanim użyje się jakiegoś oprogramowania, na osobie planującej uruchomienie danego programu spoczywa wcześniejsze zapoznanie się z warunkami użytkowania. Trzeba pamiętać, że rozliczani w tym przypadku będziemy w zgodzie z maksymą wywodzącą się z prawa rzymskiego – NIEZNAJOMOŚĆ PRAWA SZKODZI (*Ignorantia iuris nocet*).

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wyjaśnij pojęcia autorskiego prawa osobistego i autorskiego prawa majątkowego.
2. Omów zagadnienie „czas trwania praw autorskich”.
3. Na czym polega naruszenie prawa autorskiego? Jakie mogą być konsekwencje nielegalnego używania płatnego oprogramowania?
4. Wymień przykładowe elementy własności intelektualnej.
5. Gdybyś planował uzyskać patent krajowy na swój wynalazek, to do jakiego urzędu powinieneś się zwrócić?
6. Opisz podstawowe typy licencji oprogramowania.
7. Wyjaśnij różnicę między wolnym oprogramowaniem a bezpłatnym oprogramowaniem.
8. Wymień przynajmniej trzy licencje, które są związane z udostępnianiem oprogramowania bez żadnych opłat.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Przygotuj Audyt Legalności Oprogramowania na stanowisku roboczym szkolnej sali komputerowej. W tym celu:
   - A. wypisz (np. w edytorze tekstu lub arkuszu kalkulacyjnym) nazwy wszystkich programów, które są:
     - a) zainstalowane w systemie operacyjnym,
     - b) używane bezpośrednio z pliku – bez zainstalowania;
   - B. podziel programy wypisane w punkcie A na trzy grupy:
     - a) płatne,
     - b) częściowo płatne,
     - c) bezpłatne;
   - C. sprawdź, czy programy przypisane do grupy B.b. są dla instytucji edukacyjnych płatne, czy bezpłatne;
   - D. ustal dokładne nazwy licencji, które wykorzystują programy przypisane do grupy B.c. – np. XAMPP – licencja GNU General Public Licence/GPL, Visual Studio Code – licencja MIT;
   - E. na koniec przeprowadź audyt – podsumowanie, które powinno wskazywać stan prawny znaleziony na danym stanowisku roboczym.

   Przygotowany Audyt Legalności Oprogramowania dostarcz w postaci pliku do opiekuna sali komputerowej.
2. Zadanie przeznaczone do realizacji w całej szkole.
   Uczniów z klasy należy podzielić na grupy. Powinno ich być tyle, ile jest sal komputerowych w szkole. Każda grupa otrzyma przydział do danej sali komputerowej i przeprowadzi w niej Audyt Legalności Oprogramowania dla wszystkich stanowisk roboczych. Jeżeli sal jest więcej niż uczniów w klasie, to trzeba dokonać innego podziału – gwarantującego wykonanie zadania w całej szkole. Audytem można także objąć poszczególne szkolne stanowiska robocze znajdujące się poza salami komputerowymi.
   Wnioski z przeprowadzonego audytu powinny trafić do dyrekcji szkoły.
3. Przygotuj Audyt Legalności Oprogramowania dla swojego domowego stanowiska roboczego. Uwzględnij to, że w przypadku niektórych programów istnieje możliwość, iż inny będzie rodzaj licencji dla ciebie (lub twoich rodziców) jako osoby (osób) fizycznej (fizycznych), a inny dla szkoły jako instytucji prawa oświatowego.

II. Programowanie obiektowe

- Środowisko programistyczne dla obiektowych aplikacji konsolowych
- Podstawowe zasady programowania
- Typy danych w programowaniu na bazie projektu Gra edukacyjna Zgadnij Liczbę
- Ulepszenie projektu Gra edukacyjna Zgadnij Liczbę – wyrażenia, instrukcje i biblioteki
- Zasady programowania obiektowego
- Klasy i ich wykorzystanie w programowaniu
- Klasy pochodne i wzorce klas
- Programowanie wyjątków

4 Środowisko programistyczne dla obiektowych aplikacji konsolowych

ZAGADNIENIA

- Środowisko programistyczne
- Kompilator, interpreter, debugger
- Etapy kompilacji i interpretacji kodu
- Biblioteki i frameworki
- Kompilowanie i uruchamianie programów

4.1. Środowisko programistyczne – wprowadzenie

Przez wiele lat można było spotkać się ze stwierdzeniem, że programowanie to bardzo trudne zajęcie, ze względu na skalę trudności i wymagany minimalny poziom wiedzy dostępne jedynie dla nielicznych.

Zapewne wiele osób zgodzi się dziś z powyższymi sformułowaniami. Faktem jest, że programowanie nie jest łatwe, wymaga dużej wiedzy i przejścia na wysoki poziom abstrakcyjnego myślenia. Dzięki odpowiedniemu środowisku programistycznemu zawód programisty stał się jednak dostępny dla zdecydowanie większej liczby osób, niż to było w czasach tworzenia projektów programów komputerowych – na kartkach papieru, a kodów – w zwykłych notatnikach.

W tej części poruszymy kwestie dotyczące środowiska programistycznego. W skład środowiska programistycznego wchodzą:

1. komputery (tzw. stacje robocze),
2. oprogramowanie narzędziowe,
3. języki programowania wraz z właściwym środowiskiem programistycznym,
4. system operacyjny,
5. programiści.

Punktów 1., 4. i 5. nie będziemy zbyt szeroko omawiali. Należy pamiętać, że parametry komputera, czyli np. wielkość i szybkość dysku twardego, procesora, karty graficznej, RAM-u, muszą odpowiadać co najmniej minimalnym parametrom, często wskazywanym przez twórców danego oprogramowania. Podobne wymagania dotyczą używanego systemu operacyjnego.



Rys. 4.1. Elementy środowiska programistycznego

CIEKAWOSTKA

Na rys. 4.1, na którym przedstawiono środowisko programistyczne, dość ogólnie ujęto pojęcie sprzętu IT – komputery (tzw. stacje robocze). Pod tym hasłem może kryć się wiele różnego rodzaju elementów. W szczególności mogą to być:

- komputer jako skrzynka zawierająca poszczególne podzespoły, m.in. płytę główną, procesor, twardy dysk, kartę graficzną, kostki RAM;
- monitor albo – w przypadku programistów – co najmniej dwa monitory, gdyż w ten sposób proces programowania przebiega zdecydowanie efektywniej;
- infrastruktura sieciowa wraz z podłączeniem do internetu;
- urządzenia peryferyjne, np. klawiatura, myszka komputerowa, głośniki, drukarka, skaner.

W zespołach programistycznych wykorzystuje się również narzędzia, takie jak: rzutnik multimedialny, flipcharty, tablice suchościeralne.

```
dgkali5@DESKTOP-3NA7D8I: ~
dgkali5@DESKTOP-3NA7D8I:~$ sudo apt-get install hardinfo
[sudo] password for dgkali5:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  alsa-topology-conf alsa-ucm-conf at-spi2-core ca-certificates-java default-jdk-headless
  default-jre default-jre-headless fonts-dejavu-extra java-common libasound2
  libasound2-data libatk-bridge2.0-0 libatk-wrapper-java libatk-wrapper-java-jni
  libatspi2.0-0 libdrm-amdgpu1 libdrm-common libdrm-intel1 libdrm-nouveau2 libdrm-radeon1
  libdrm2 libfontenc1 libgif7 libgl1 libgl1-mesa-dri libglapi-mesa libglvnd0 libglx-mesa0
  libglx0 libice-dev libice6 liblcms2-2 libllvm11 libnspr4 libnss3 libpciaccess0
  libpcsclite1 libpthread-stubs0-dev libsm-dev libsm6 libvulkan1 libwayland-client0
  libx11-dev libx11-xcb1 libxau-dev libxaw7 libxcb-dri2-0 libxcb-dri3-0 libxcb-glx0
  libxcb-present0 libxcb-randr0 libxcb-shape0 libxcb-sync1 libxcb-xfixes0 libxcb1-dev
  libxdmcp-dev libxft2 libxkbfile1 libxmu6 libxshmfence1 libxt-dev libxt6 libxtst6 libxv1
  libxxf86dga1 libxxf86vm1 libz3-4 mesa-vulkan-drivers openjdk-11-jdk
  openjdk-11-jdk-headless openjdk-11-jre openjdk-11-jre-headless x11-common x11-utils
  x11proto-dev xorg-sgml-doctools xtrans-dev
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  adwaita-icon-theme fontconfig gtk-update-icon-cache hicolor-icon-theme libcairo-gobject2
  libcairo2 libdatrie1 libfribidi0 libgail-common libgail18 libgdk-pixbuf-2.0-0
  libgdk-pixbuf-xlib-2.0-0 libgdk-pixbuf2.0-0 libgdk-pixbuf2.0-bin libgdk-pixbuf2.0-common
  libgtk2.0-0 libgtk2.0-bin libgtk2.0-common libpango-1.0-0 libpangocairo-1.0-0
  libpangoft2-1.0-0 libpci3 libpixman-1-0 librsvg2-2 librsvg2-common libthai-data libthai0
  libxcb-render0 libxcursor1 lm-sensors pci.ids pciutils
Suggested packages:
  mesa-utils gvfs librsvg2-bin fancontrol read-edid i2c-tools
The following NEW packages will be installed:
```

Rys. 4.2. Instalacja w systemie Linux aplikacji hardinfo pokazującej parametry systemu komputerowego

Przykładowe minimalne wymagania sprzętowe podczas pracy na programie:

- **Visual Studio Community 2019**[1] to:
  - procesor 1,8 GHz lub szybszy (zalecany czterordzeniowy lub lepszy);
  - 2 GB pamięci RAM; zalecane 8 GB pamięci RAM (co najmniej 2,5 GB w przypadku uruchamiania na maszynie wirtualnej);
  - miejsce na dysku twardym: co najmniej 800 MB, maksymalnie 210 GB dostępnego miejsca (w zależności od instalowanych funkcji); typowe instalacje wymagają 20–50 GB wolnego miejsca;
  - szybkość dysku twardego: aby zwiększyć wydajność, system Windows i program Visual Studio należy zainstalować na dysku półprzewodnikowym (SSD);
  - karta wideo obsługująca rozdzielczość ekranu co najmniej 720p (1280 × 720); program Visual Studio będzie działał najlepiej w rozdzielczości WXGA (1366 × 768) lub wyższej.

- **Android Studio 2019**[2]

  System Windows:
  - Microsoft® Windows® 7/8/10 (64-bitowy);
  - co najmniej 4 GB pamięci RAM, zalecane 8 GB pamięci RAM;
  - co najmniej 2 GB dostępnego miejsca na dysku, 4 GB zalecane (500 MB dla IDE + 1,5 GB dla Android SDK i obraz systemu emulatora);
  - minimalna rozdzielczość ekranu 1280 × 800;

  System Linux:
  - pulpit GNOME lub KDE;
  - testowany na gLinux na bazie Debiana 64-bitowy rozkład umożliwiający uruchamianie aplikacji 32-bitowych;
  - biblioteka GNU C (glibc) 2.19 lub nowsza;
  - co najmniej 4 GB pamięci RAM, zalecane 8 GB pamięci RAM;
  - co najmniej 2 GB dostępnego miejsca na dysku, 4 GB zalecane (500 MB dla IDE + 1,5 GB dla Android SDK i obraz systemu emulatora);
  - minimalna rozdzielczość ekranu 1280 × 800.

[1] Źródło: https://docs.microsoft.com/pl-pl/visualstudio/releases/2019/system-requirements (dostęp: 8.03.2021).
[2] Źródło: https://developer.android.com/studio (dostęp: 8.03.2021).

CIEKAWOSTKA

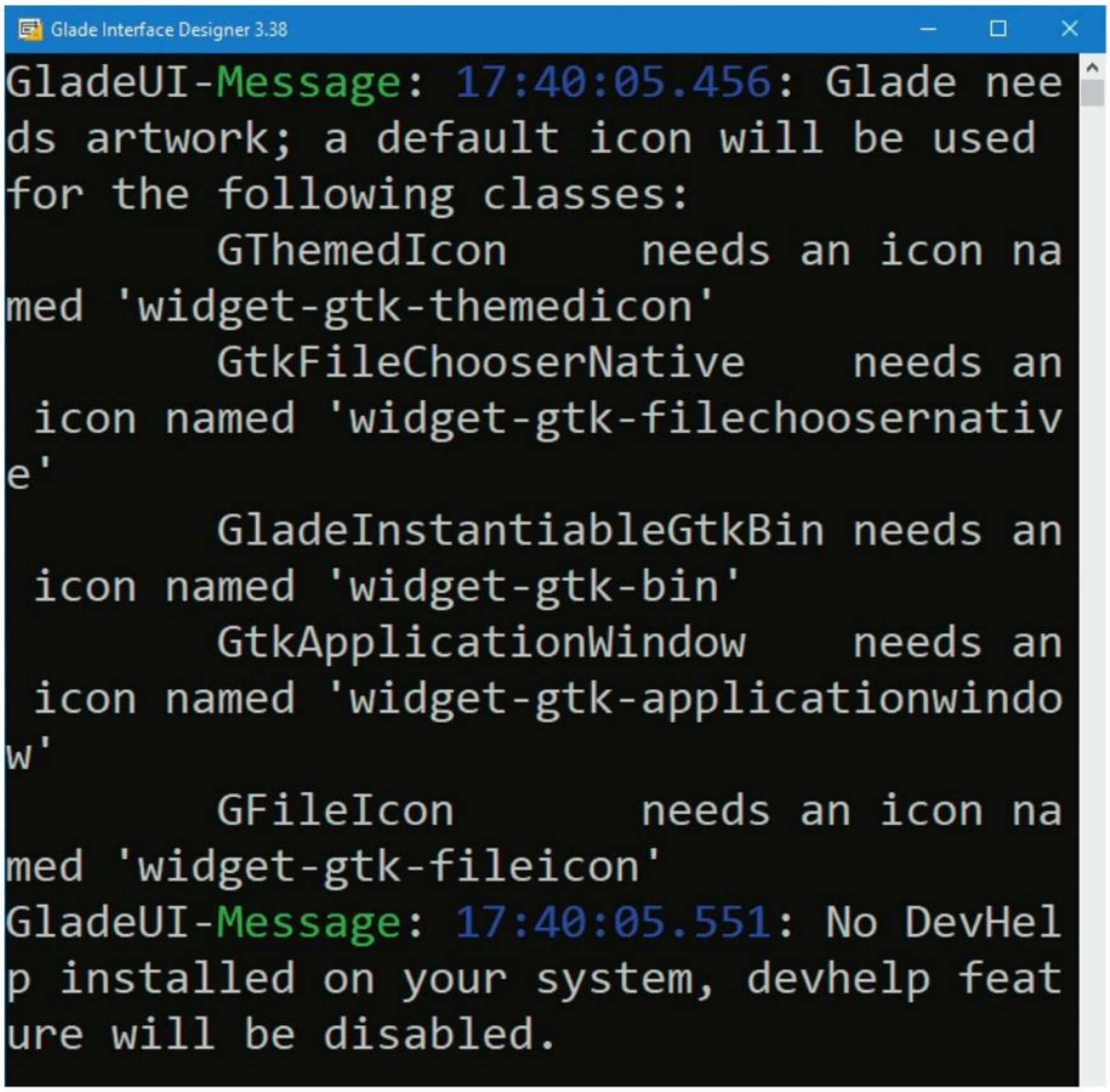
W każdym systemie operacyjnym jesteśmy w stanie sprawdzić parametry sprzętowe używanego zestawu komputerowego. W systemie Linux instaluje się np. aplikację hardinfo – w konsoli – poleceniem (w dystrybucjach opartych na Debianie):

`sudo apt-get install hardinfo` (rys. 4.2).

Po instalacji możemy wywołać program hardinfo przez zwykłe wpisanie jego nazwy do konsoli systemowej. Efektem są dziesiątki linijek informacji, które pokazują parametry sprzętu komputerowego, na którym wskazany program został uruchomiony. Ich niewielki wycinek – jako przykład – przedstawiono na rys. 4.3.

```
dg@DESKTOP-3NA7D8I: ~
Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz        8x 2801.00 MHz  0.98
Intel(R) Pentium(R) DualCPUT3200@ 2.00GHz        2x 2000.00 MHz  5.50
Intel(R) Celeron(R) CPU 2.53GHz 1x 2527.00 MHz  10.12
Genuine Intel(R) CPU T1350@ 1.86GHz     1x 1867.00 MHz  83.47
AMD Turion(tm) 64 X2 Mobile Technology TL-60     2x 2000.00 MHz  8.22
AMD Turion(tm) X2 Dual-Core Mobile RM-70         2x 500.00 MHz   8.31
Intel(R) Core(TM) Duo CPUT2350@ 1.86GHz 2x 1867.00 MHz  5.36
Intel(R) Core(TM)2 Duo CPU T5750@ 2.00GHz        2x 1994.00 MHz  4.23
Intel(R) Core(TM)2 Duo CPU T9500@ 2.60GHz        2x 2593.00 MHz  3.12
AMD Athlon(tm) X2 Dual-Core QL-60        2x 1900.00 MHz  9.08
Intel(R) Core(TM)2 Duo CPU T5250@ 1.50GHz        2x 1500.00 MHz  5.51
Mobile AMD Athlon(tm) XP-M Processor 3000+       1x 799.00 MHz   43.15
AMD Athlon(tm) 64 X2 Dual Core Processor 4000+   2x 1000.00 MHz  8.00
AMD Athlon(tm) XP processor 1800+        1x 1533.00 MHz  25.14
Genuine Intel(R) CPU2140@ 1.60GHz        2x 1595.00 MHz  5.44
Intel(R) Core(TM)2 Quad CPUQ6600@ 2.40GHz        4x 2400.00 MHz  1.63
Intel(R) Core(TM)2 Duo CPU P7350@ 2.00GHz        2x 2000.00 MHz  4.76
Genuine Intel(R) CPU T2080@ 1.73GHz      2x 1730.00 MHz  4.99
Intel(R) Celeron(R) CPU 2.50GHz 1x 2492.00 MHz  14.46
Intel(R) Pentium(R) M processor 1.86GHz 1x 1867.00 MHz  11.88
Intel(R) Core(TM)2 CPU T5200@ 1.60GHz    2x 1600.00 MHz  6.49
Intel(R) Core(TM)2 CPU6320@ 1.86GHz      2x 1860.00 MHz  4.91
AMD Athlon(tm) XP        1x 906.00 MHz   84.35
AMD Athlon(tm) 64 Processor 3000+        1x 1800.00 MHz  25.19
Intel(R) Pentium(R) 4 CPU 2.60GHz        2x 2600.00 MHz  8.54
Intel(R) Core(TM)2 Quad CPUQ9400@ 2.66GHz        4x 2660.00 MHz  1.49
Intel(R) Pentium(R) M processor 1600MHz 1x 1600.00 MHz  13.50
Pentium(R) Dual-Core CPU T4200@ 2.00GHz 2x 2000.00 MHz  4.62
Intel(R) Core(TM)2 Quad CPU @ 2.40GHz    4x 2400.00 MHz  1.70
Intel(R) Core(TM)2 Quad CPUQ8300@ 2.50GHz        4x 2497.00 MHz  1.66
Intel(R) Core(TM)2 Extreme CPU X7900@ 2.80GHz    2x 2800.00 MHz  3.06
Intel(R) Celeron(R) D CPU 3.20GHz        1x 3196.00 MHz  8.55
AMD Sempron(tm) 2600+    1x 1840.00 MHz  20.95
AMD Athlon(tm) 64 X2 Dual Core Processor 4400+   2x 1000.00 MHz  6.80
Intel(R) Celeron(R) CPU 2.66GHz 1x 2666.00 MHz  9.89
Intel(R) Core(TM) Duo CPUT2450@ 2.00GHz 2x 2000.00 MHz  4.33
Intel(R) Celeron(R) M CPU520@ 1.60GHz    1x 1600.00 MHz  11.35
```

Rys. 4.3. Użycie programu hardinfo w konsoli Linux Ubuntu – efektem są dziesiątki linijek informacji o parametrach sprzętu komputerowego



Rys. 4.4. Glade Gnome – widok konsoli w trakcie uruchamiania programu

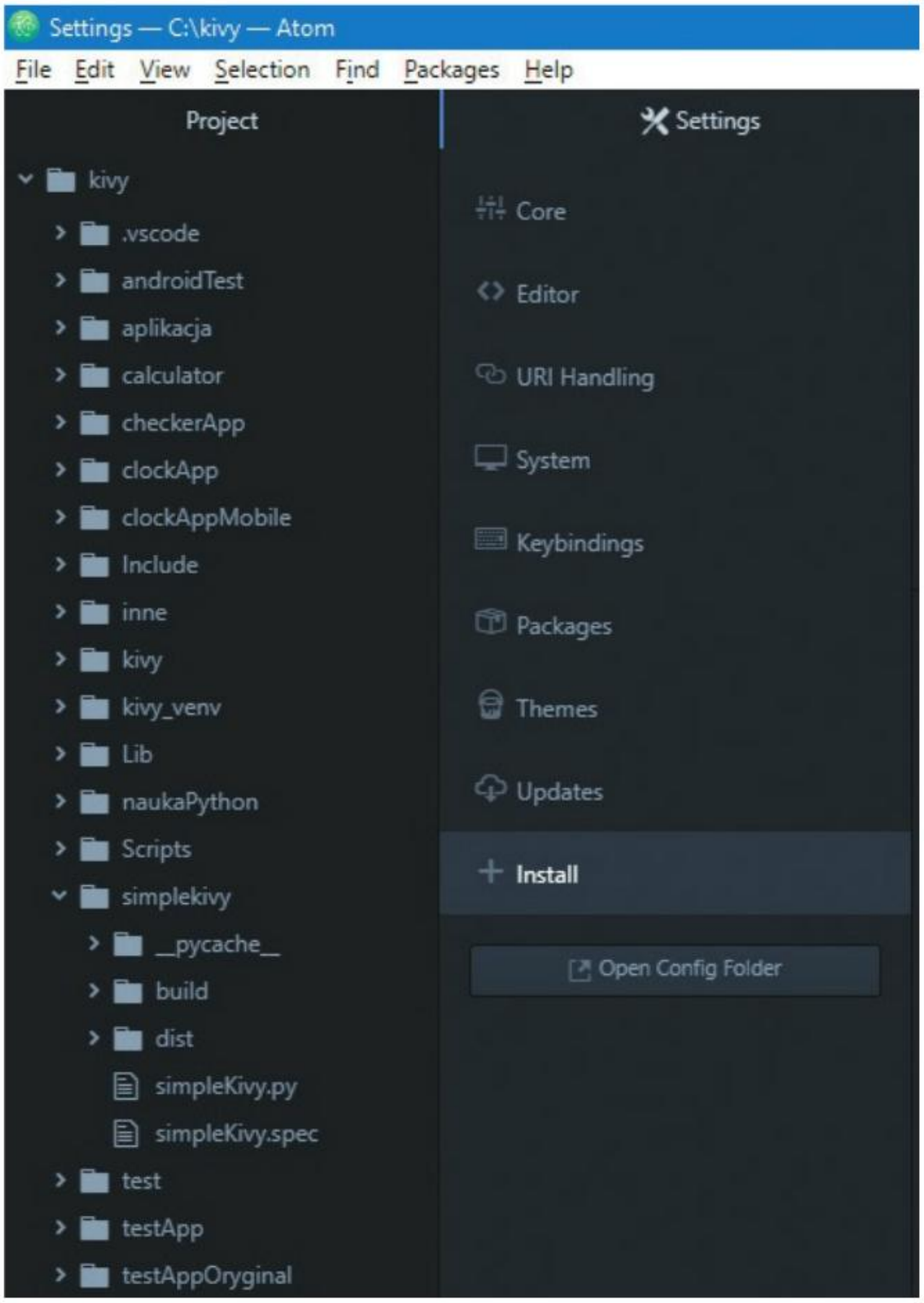
Aby programować w określonym języku i używać danego narzędzia projektowego, trzeba także dysponować określonym systemem operacyjnym, a w nim – niezbędnymi zależnościami (bibliotekami, modułami, puginami itp.) dla danej aplikacji. Przykładowo: aby używać narzędzia (tzw. RAD) GLAD GNOME (https://glade.gnome.org/ – na rys. 4.4 przedstawiono uruchomienie tego programu), trzeba mieć w systemie zainstalowane m.in. zależności (źródło: https://packages.msys2.org/package/mingw-w64-x86_64-glade):

- ingw-w64-x86_64-adwaita-icon-theme,
- mingw-w64-x86_64-glib-networking,
- mingw-w64-x86_64-gtk-update-icon-cache,
- mingw-w64-x86_64-gtk3,
- mingw-w64-x86_64-libxml2,
- mingw-w64-x86_64-python (Python widgets support),
- mingw-w64-x86_64-gcc,
- mingw-w64-x86_64-gnome-common,
- mingw-w64-x86_64-gobject-introspection,

- mingw-w64-x86_64-gtk-doc,
- mingw-w64-x86_64-itstool,
- mingw-w64-x86_64-meson,
- mingw-w64-x86_64-ninja,
- mingw-w64-x86_64-pkg-config,
- mingw-w64-x86_64-python-gobject.

CIEKAWOSTKA

W wielu programach podstawowa funkcjonalność może zostać wielokrotnie zwiększona przez instalację dodatkowych modułów. Przykładowo: w aplikacji Atom.js w sekcji Settings trzeba kliknąć ikonkę Install, aby pojawił się panel umożliwiający dostęp do dodatkowych modułów.



Rys. 4.5. Widok aplikacji Atom.js – kliknięcie ikonki Install w sekcji Settings umożliwi instalację dodatkowych modułów, które pozwolą na uzyskanie rozszerzonej funkcjonalności w stosunku do standardowej wersji tego programu

Nie będziemy wskazywali, jakie konkretnie do danych języka programowania i narzędzia informatycznego powinny być:

- sprzęt,
- system operacyjny,
- zainstalowane zależności (dodatki, moduły, biblioteki itp. – nazwy tego typu elementów bywają różne, zależnie np. od dokładnej funkcjonalności, środowiska – miejsca wykorzystania).

To zadanie należy do osób używających danego środowiska. Muszą one zdobyć niezbędne informacje i dostosować środowisko pracy do wytycznych przedstawionych przez producentów i/lub twórców narzędzi informatycznych.

Poniżej znajduje się przykładowa lista środowisk do tworzenia i uruchamiania aplikacji. Oczywiście to tylko kilka możliwych rozwiązań, w realnym programowaniu jest ich o wiele więcej.

Aplikacje desktopowe

Dla C++ / C#

Glade Gnome, CodeBlocks, Visual Studio Community, Xcode, Visual Studio Code, Qt, Codelite, Kite, Emacs.

Dla Python

Tkinter, Idle, CPython, pyCharm, PyQt, PyInstaller, Visual Studio Code, Atom, PyCharm, Spyder, Jupyter, Eclipse.

Dla JAVA

IntellijIDEA, Swing,Eclipse,Qt.

Dla JavaScript

Atom, Electron, node.js, Visual Studio Code.

Język programowania JavaScript jest dostępny prawie wszędzie. Jeszcze kilka lat temu mogło się wydawać czymś nieprawdopodobnym, żeby za pomocą tego języka programowania tworzyć aplikacje działające bezpośrednio w systemie operacyjnym Windows. Zmieniło się to dzięki niesamowitym możliwościom, jakie daje twórcom aplikacji framework Electron (oficjalna strona projektu: https://www.electronjs.org/), o którego rozwój, poza społecznością internetową, dba organizacja OpenJS Foundation (oficjalna strona projektu: https://openjsf.org/).

Co ciekawe, na jej stronie internetowej można znaleźć m.in. wykaz wszystkich projektów *open source*, które są przez nią rozwijane. Do najbardziej znanych i najczęściej wykorzystywanych, nie tylko przez deweloperów języka programowania JavaScript, należą m.in.: Dojo, Electron, ESLint, Grunt, jQuery, jQuery Mobile, jQuery UI, Lodash, Mocha, NativeScript, Node.js, webpack.

CIEKAWOSTKA

Jednym z najbardziej popularnych sposobów tworzenia aplikacji mobilnych jest React-Native. Umożliwia on stworzenie zaawansowanych programów na większość dostępnych systemów mobilnych. Plikiem, który zarządza zależnościami w projekcie ReactNative, jest package.json – jego przykładowe ustawienia przedstawiono na rys. 4.6.

package.json

```
{
  "name": "Projekt Mobilny w React Native",
  "version": "0.0.2",
  "scripts": {
    "start": "react-native start",
    "test": "jest",
    "lint": "eslint ."
  },
  "dependencies": {
    "react": "16.8.6",
    "react-native": "0.60.4"
  },
  "devDependencies": {
    "@babel/core": "^7.5.5",
    "@babel/runtime": "^7.5.5",
    "@react-native-community/eslint-config": "^0.0.5",
    "babel-jest": "^24.8.0",
    "eslint": "^6.1.0",
    "jest": "^24.8.0",
    "metro-react-native-babel-preset": "^0.55.0",
    "react-test-renderer": "16.8.6"
  },
  "jest": {
    "preset": "react-native"
  }
}
```

Rys. 4.6. Framework ReactNative – widok przykładowego kodu pliku konfiguracyjnego package.json

Aplikacje mobilne

Dla Python

Atom, Visual Studio Code, Kivy, BeeWare.

Dla C++ / C# / C-Object

Visual Studio Community, Xamarin, Android Studio, Xcode, Glade Gnome, Visual Studio for Mac.

Dla JavaScript

Atom, node.js, ReactNative, Android Studio, Expo, Apache Cordova, Ionic, NativeScripts, Framework7 (i co najmniej kilkadziesiąt innych – w świecie JS mamy do czynienia z bogactwem urodzaju).

```
using Xamarin.Forms;
using DGStackLayout.Services;
using DGStackLayout.Views;

namespace DGStackLayout
{
    4 references
    public partial class App : Application
    {

        2 references
        public App()
        {
            InitializeComponent();
```

Rys. 4.7. Przykład kodu języka C# z aplikacji opartej na frameworku Xamarin, służącym do tworzenia aplikacji mobilnych

CIEKAWOSTKA

Oprócz wspomnianego ReactNative kolejnym wartym odnotowania frameworkiem do tworzenia aplikacji mobilnych jest Xamarin – w połączeniu z:

- językiem programowania C#,
- XAML – językiem opisu interfejsu użytkownika.

Xamarin umożliwia stworzenie zaawansowanych programów na większość dostępnych systemów mobilnych. Na rys. 4.7 znajdziesz przykład kodu wymienionych technologii.

Więcej informacji o tych technologiach uzyskasz na stronach:

Xamarin
https://dotnet.microsoft.com/apps/xamarin

C#
https://docs.microsoft.com/en-us/dotnet/csharp/

XAML
https://docs.microsoft.com/pl-pl/dotnet/desktop/wpf/fundamentals/xaml?view=netdesktop-5.0

Aplikacje webowe

Dla Python
Django, Atom, Visual Studio Code, Glade Gnome.

Dla C#
ASP.NET, Visual Studio Community, Visual Studio Code, Atom, Glade Gnome.

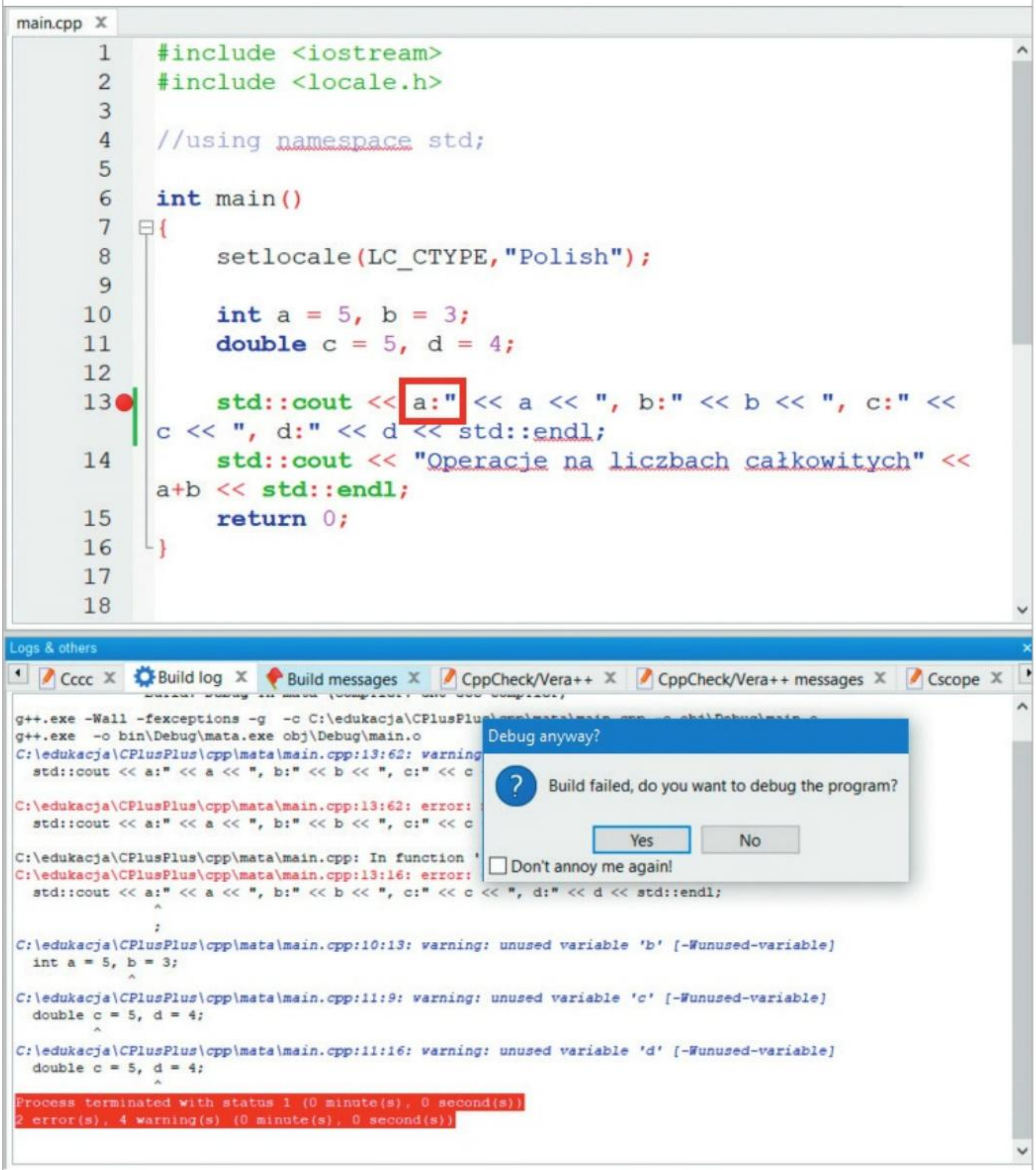
Dla JavaScript
React, Angular, Vue, Svelte, Redux, Ember, Meteor, Mithril, Aurelia, Next.js, Backbone, node.js, Atom, Visual Studio Code, Brackets.

Dla PHP
Symfony, Laravel, CodeIgniter, Zend Framework / Laminas Project, Yii, CakePHP, Phalcon, FuelPHP, Fat-Free Framework, NetBeans, Eclipse, Visual Studio Code.

W przypadku zainteresowania bogactwem i różnorodnością poszczególnych środowisk informatycznych polecamy poszukanie odpowiednich informacji w internecie.

4.2. Kompilator, interpreter, debugger

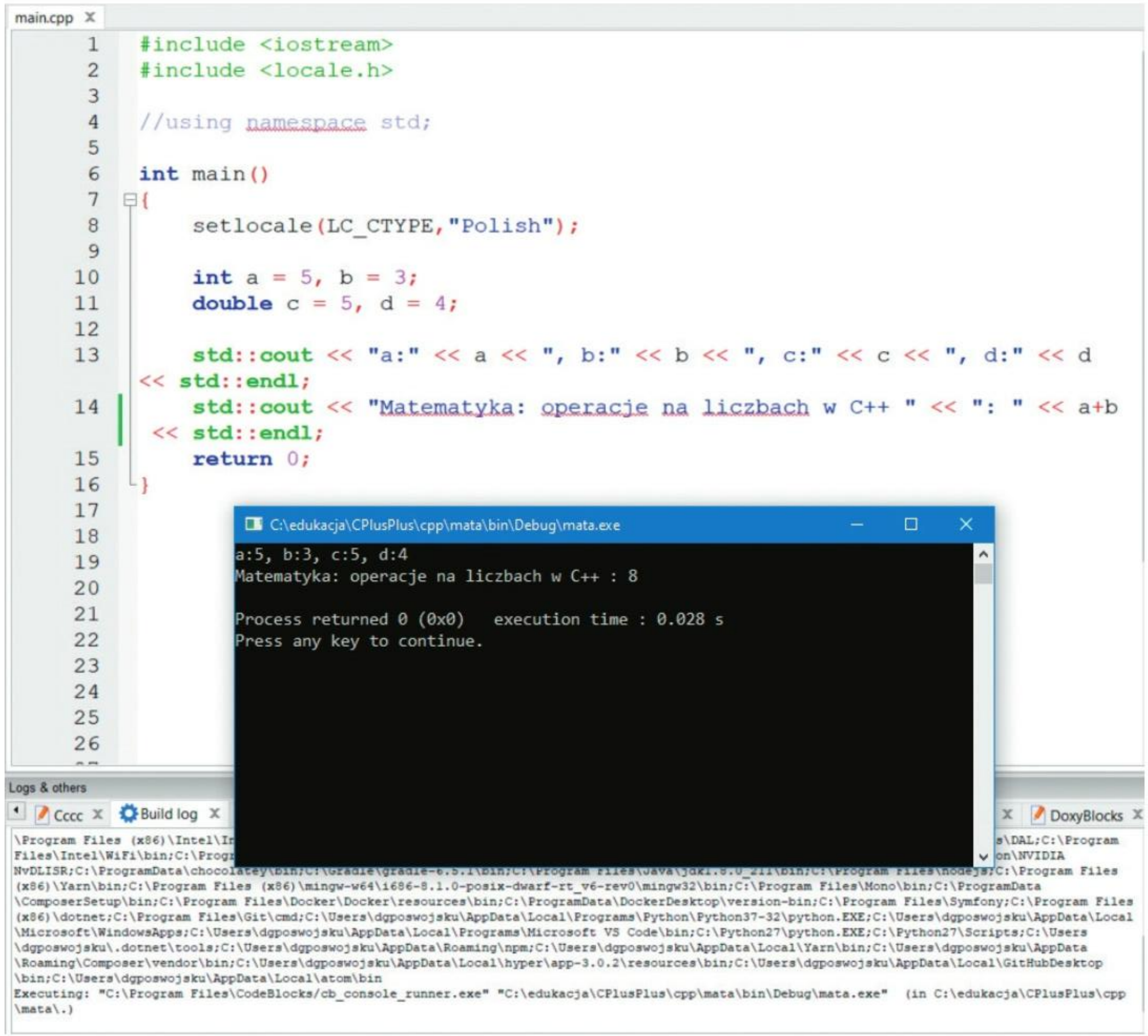
Zajmiemy się teraz teorią technologii programistycznych. Na początek informacja dotycząca rodzajów języków programowania. Kryteriów podziału w tym obszarze jest wiele. Skupimy się na jednym: tłumaczenie kodu źródłowego danego języka na kod maszynowy.



Rys. 4.8. Przykład błędu w kodzie (brak cudzysłowu początkowego – zaznaczony kolorem) języka C++, środowisko CodeBlocks

Języki kompilowane

Kod źródłowy jest tłumaczony na kod maszynowy (czyli do postaci binarnej), a następnie zapisywany jako aplikacja w danym systemie operacyjnym (np. pliki .exe w systemie Windows). Aby to lepiej zrozumieć, dodajmy, że **kompilacja** jest terminem używanym w procesie tłumaczenia z języka wyższego poziomu na język niższego poziomu, a proces odwrotny to **dekompilacja**. Przykłady języków tak działających to m.in.: C, C++, C#, Java.

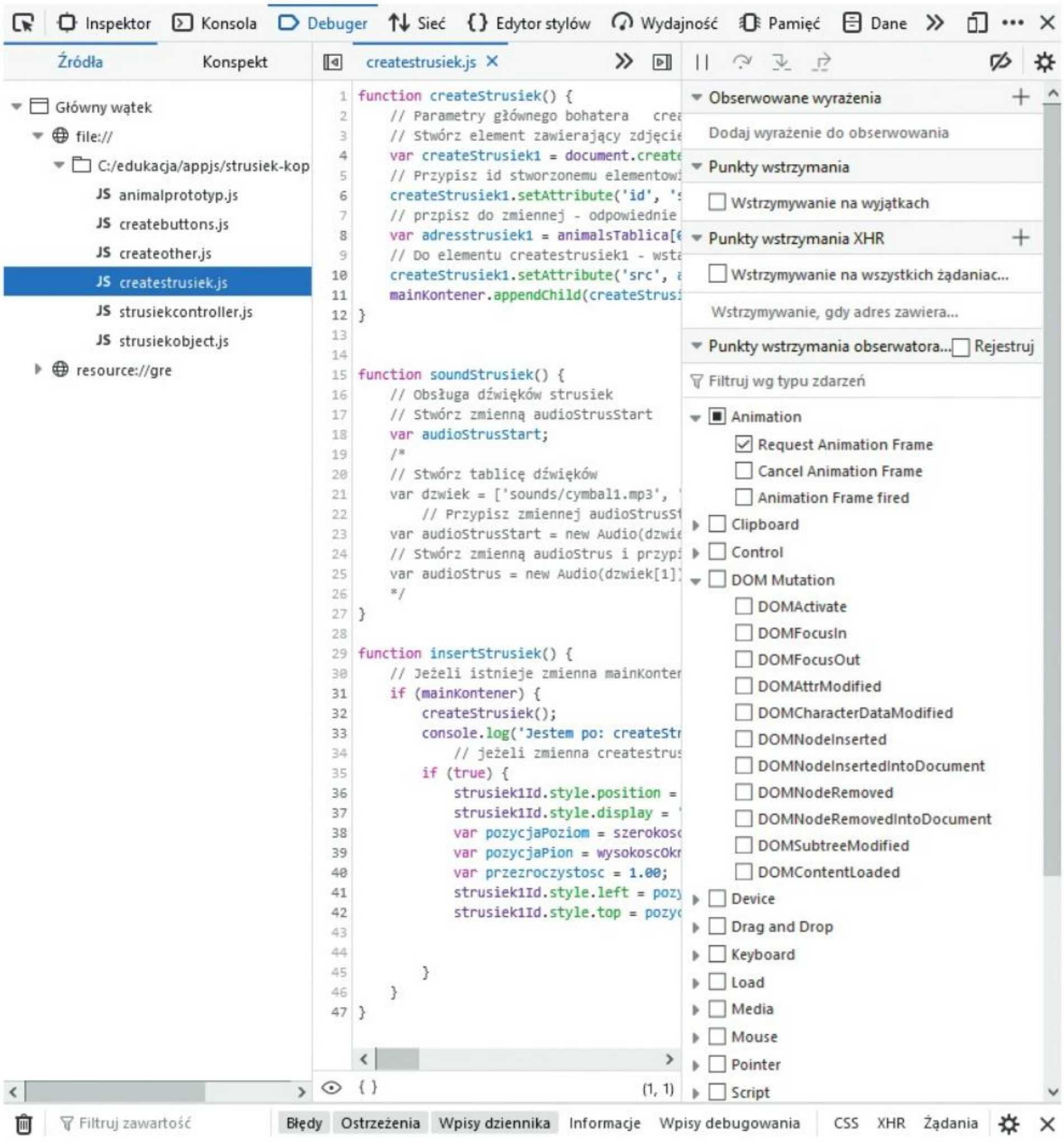


Rys. 4.9. Przykład udanej kompilacji kodu stworzonego w języku programowania C++ – środowisko CodeBlocks (efekt działania kodu znajduje się w konsoli CMD)

Języki interpretowane

Interpreter tłumaczy kod źródłowy na kod maszynowy – instrukcja po instrukcji. Na rys. 4.10 przedstawiono przykład kodu JavaScript. Oprócz JavaScript językami interpretowanymi są m.in.: PHP, Python, Bash, Perl.

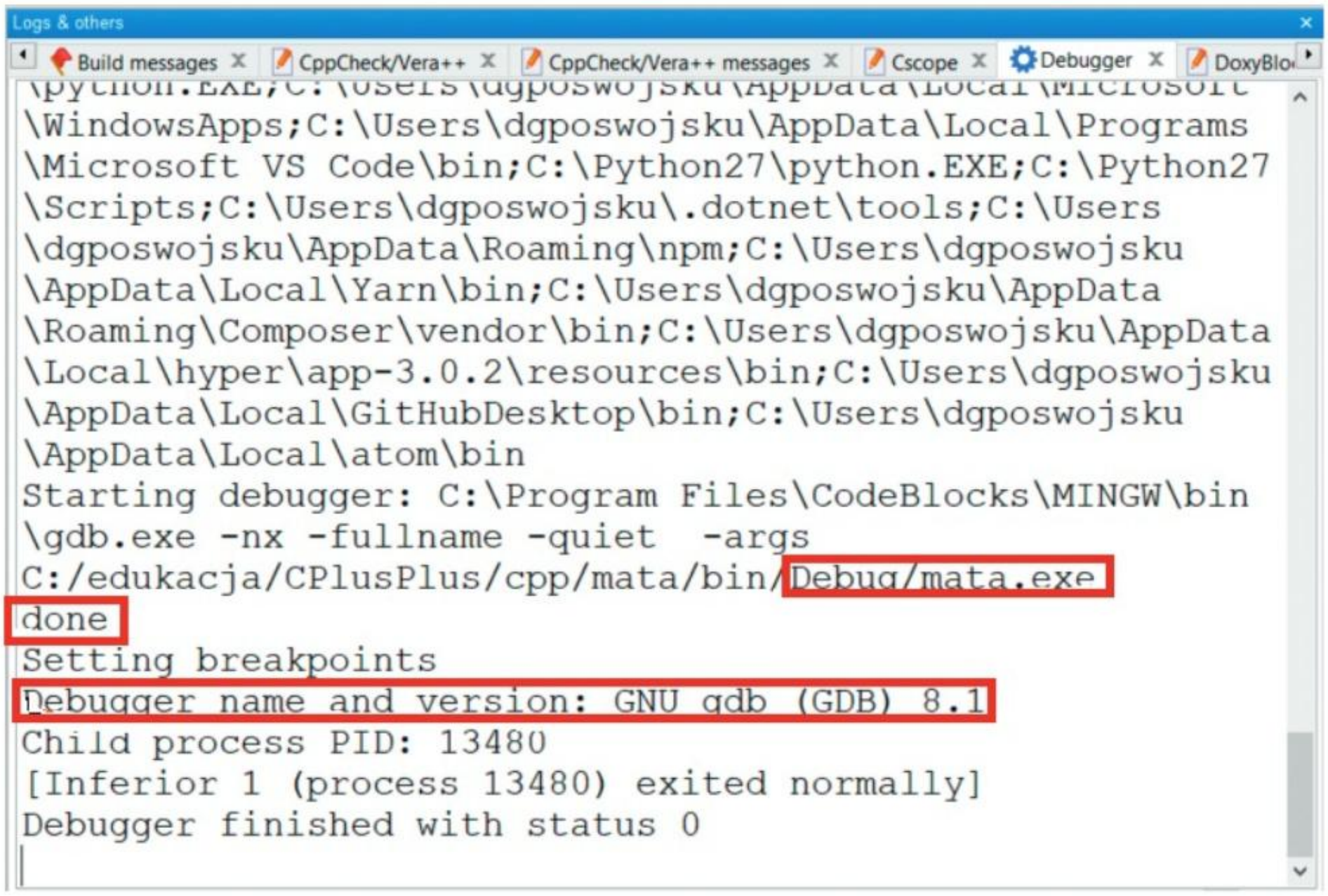
W informatyce program komputerowy, który tłumaczy kod źródłowy na kod maszynowy, nazywa się **kompilatorem**. **Interpreter** to program komputerowy, który wykonuje inne programy.




```
1  function createStrusiek() {
2      // Parametry głównego bohatera   crea
3      // Stwórz element zawierający zdjęcie
4      var createStrusiek1 = document.create
5      // Przypisz id stworzonemu elementowi
6      createStrusiek1.setAttribute('id', 's
7      // przpisz do zmiennej - odpowiednie
8      var adresstrusiek1 = animalsTablica[6
9      // Do elementu createstrusiek1 - wsta
10     createStrusiek1.setAttribute('src', a
11     mainKontener.appendChild(createStrusi
12 }
13
14
15 function soundStrusiek() {
16     // Obsługa dźwięków strusiek
17     // Stwórz zmienną audioStrusStart
18     var audioStrusStart;
19     /*
20     // Stwórz tablicę dźwięków
21     var dzwiek = ['sounds/cymbal1.mp3', '
22         // Przypisz zmiennej audioStrussi
23     var audioStrusStart = new Audio(dzwie
24     // Stwórz zmienną audioStrus i przypi
25     var audioStrus = new Audio(dzwiek[1])
26     */
27 }
28
29 function insertStrusiek() {
30     // Jeżeli istnieje zmienna mainKonter
31     if (mainKontener) {
32         createStrusiek();
33         console.log('Jestem po: createStr
34             // jeżeli zmienna createstrus
35         if (true) {
36             strusiek1Id.style.position =
37             strusiek1Id.style.display = '
38             var pozycjaPoziom = szerokosc
39             var pozycjaPion = wysokoscOkn
40             var przezroczystosc = 1.00;
41             strusiek1Id.style.left = pozy
42             strusiek1Id.style.top = pozyc
43
44
45         }
46     }
47 }
```

Rys. 4.10. Debugger kodu języka JavaScript zawarty w panelu dewelopera przeglądarki internetowej FireFox

Gdy już napiszemy kod, np. w języku C++, to chcielibyśmy zobaczyć jego działanie. Jeżeli w stworzonym dziele jest błąd, to nie dojdzie do uruchomienia napisanego programu. Czasem próba włączenia błędnie napisanego kodu może zawiesić system operacyjny lub inny program (np. edytor/IDE, w którym pracujemy nad kodem). Lepszym rozwiązaniem, zwłaszcza przy rozbudowanych strukturach kodu, będzie najpierw sprawdzenie jego poprawności, a dopiero potem – gdy nie będzie w nim błędów – uruchomienie programu. Proces nadzorowania wykonania programu wraz ze znalezieniem w nim błędów określa się jako **debugowanie**, a program, który to wykonuje – **debuggerem** (rys. 4.11).



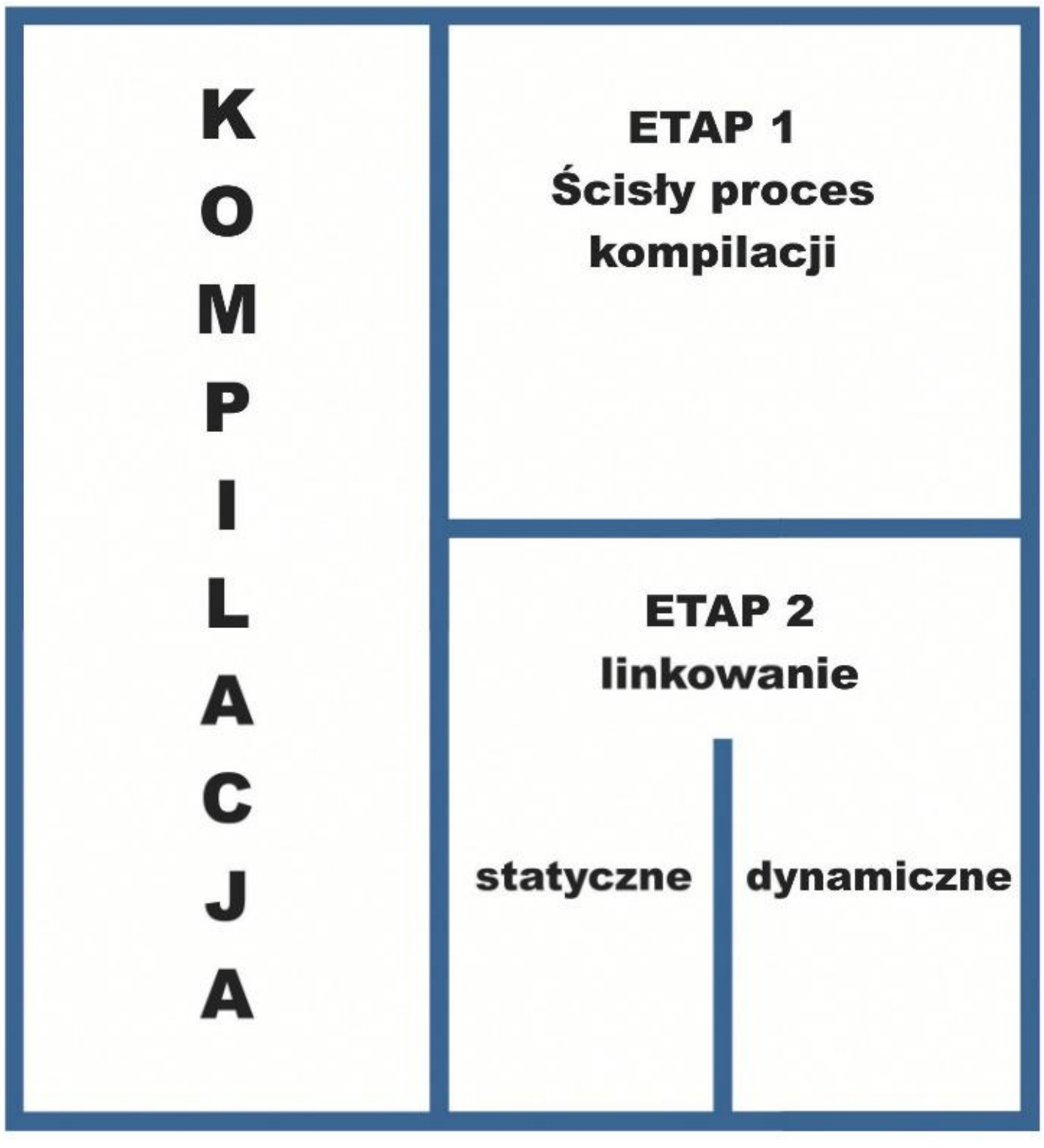
Rys. 4.11. Przykład debugowania kodu stworzonego w języku programowania C++ – środowisko CodeBlocks

4.3. Etapy kompilacji i interpretacji kodu

W poprzedniej części opisywaliśmy proces kompilacji kodu. Aby wyjaśnić różnice między kompilowaniem, interpretacją i debugowaniem, nieco uprościliśmy pewne aspekty części z tych pojęć. Szczególnie dotyczy to kompilacji kodu. Jest to tłumaczenie kodu źródłowego na maszynowy, ale w rzeczywistości jest to proces dużo bardziej skomplikowany. Przyjmijmy, że kompilacja składa się z dwóch etapów:

- etap 1:
  - ścisły proces kompilacji;
- etap 2:
  - linkowanie (ten etap specjalny wykonuje program – linker):
    - statyczne – linker łączy kody obiektowe plików źródłowych z binariami bibliotek rozszerzających (np. pliki .lib), które są już gotowe do działania,
    - dynamiczne – linker łączy kody obiektowe plików źródłowych z binariami bibliotek (np. pliki .dll), ale robi to dopiero w trakcie wykonywania aplikacji.

Czasem można spotkać się z opinią, że linkowanie i kompilacja to dwa odrębne procesy. Także dla takiego podejścia można znaleźć uzasadnienie. Istotne jest jednak to, że po ścisłym procesie kompilacji następuje proces linkowania, wykonywany przez program określany mianem linkera. Omawiane zagadnienie przedstawiono na rys. 4.12.

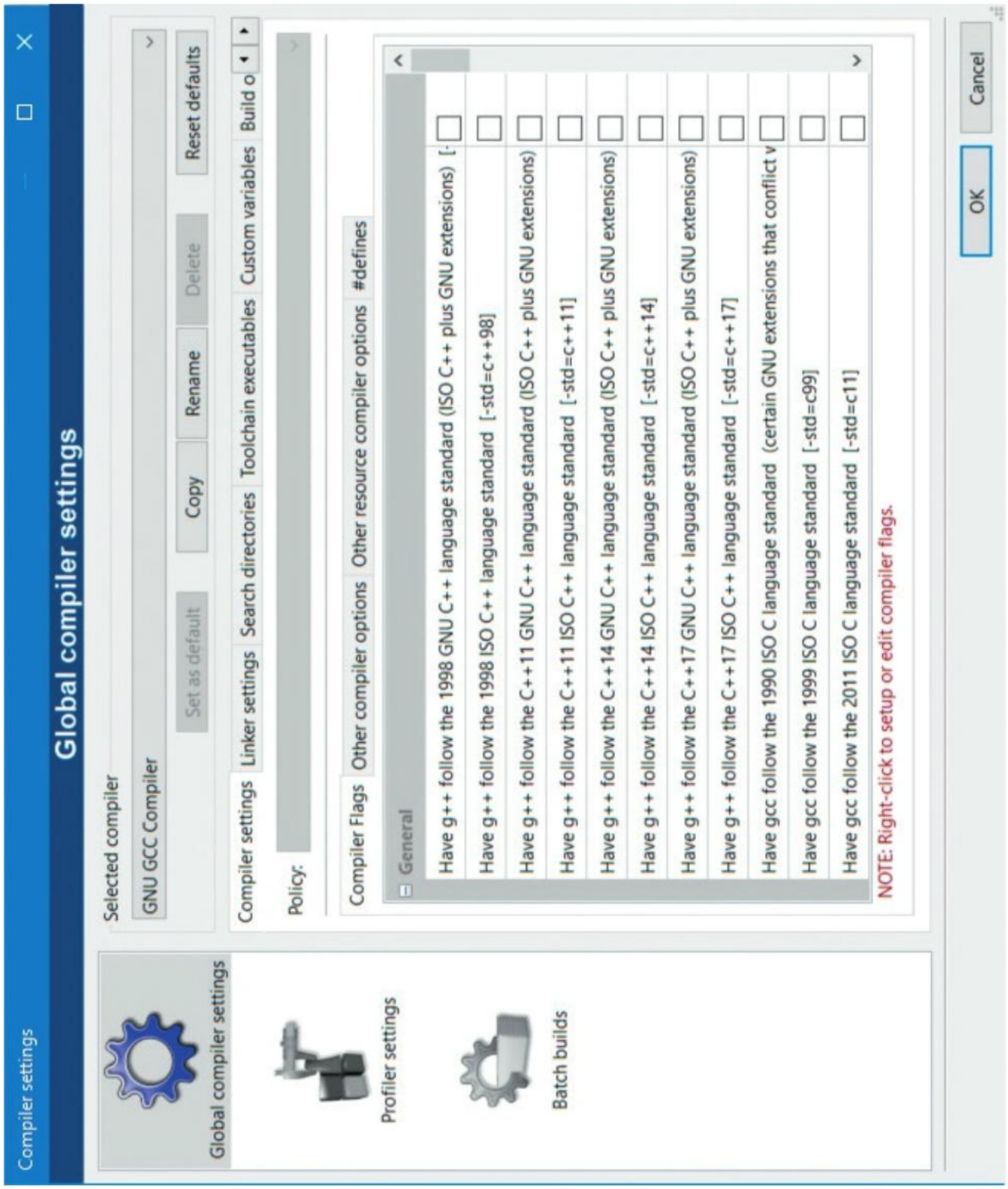


Rys. 4.12. Grafika poglądowa: proces kompilacji

Do powyższych opisów kompilacji można dodać jeszcze co najmniej jeden element. Warto wiedzieć, że ścisła kompilacja dzieli się na fazy. Zwykle wyodrębnia się dwie:

- odczyt i analiza (leksykalna, składniowa, semantyczna) kodu źródłowego przeprowadzane na podstawie reguł danego języka;
- generowanie kodu wynikowego z uwzględnieniem jakiegoś priorytetu, np. jak najmniejsza objętość, maksymalna wydajność.

Na rys. 4.13 można zobaczyć ustawienia programu CodeBlocks w zakresie kompilatorów dostępnych i używanych. Dzięki tej opcji wskazana aplikacja jest wysoce konfigurowalna i może skorzystać z wielu kompilatorów.



Rys. 4.13. CodeBlocks – widok ustawień wykorzystywanego kompilatora (główne menu – Settings – Compiler)

Interpretacja kodu odbywa się linia po linii. Interpreter przesuwa się po kolejnych linijkach zapisanego kodu, interpretując go w celu jego uruchomienia.

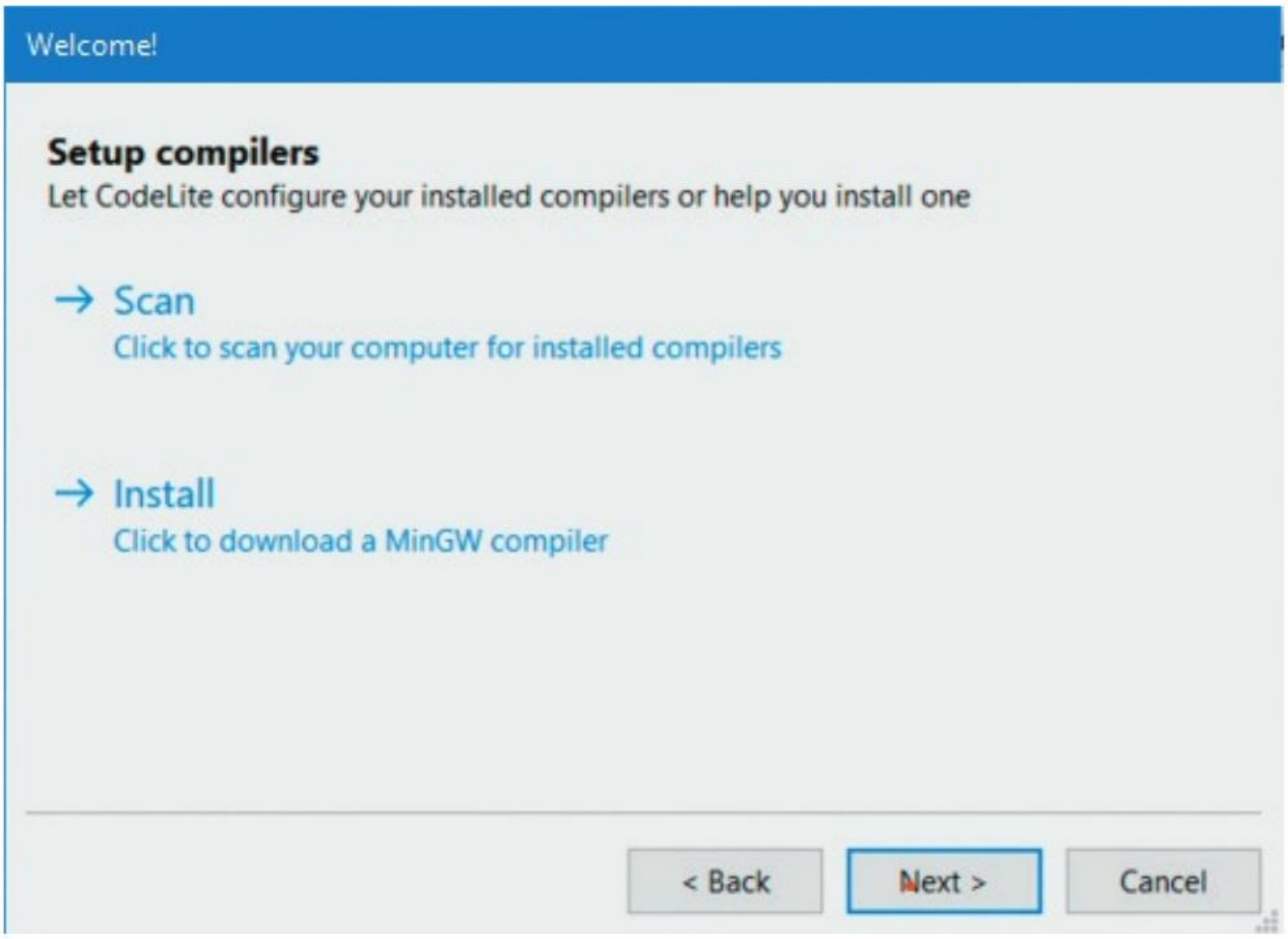
Jeżeli napotka problem, bardzo często całość danego rozwiązania się zawiesza. Przykładem mogą być błędy w języku programowania JavaScript na stronach internetowych. Bardzo często po znalezieniu błędu przez silnik przeglądarki następuje „zwieszenie" całego projektu webowego.

Jest jeszcze jeden istotny aspekt porównania interpretacji kodu z kompilacją w procesie programowania – **szybkość działania**. Wykonanie programu za pomocą interpretera jest wolniejsze niż wykonanie kodu skompilowanego. Zajmuje także więcej zasobów systemowych, co przy wielkich projektach ma oczywiście duże znaczenie. Interpretacja kodu ma również zalety, szczególnie w zakresie czasu niezbędnego do zaangażowania programisty. Zwykle zajmuje ona relatywnie mniej czasu niż kompilacja i uruchomienie. Dzieje się tak ze względu na długość procesu (cyklu) programowania. Zauważ, że podczas tworzenia i testowania kodu dla:

- interpretacji mamy proces: edycja – interpretacja – debugowanie;
- kompilacji mamy proces: edycja – kompilacja – debugowanie – uruchomienie.
  Cały proces wydłuża się w przypadku procesów związanych z językami kompilowanymi.

CIEKAWOSTKA

Program CodeLight to bardzo dobre rozwiązanie typu IDE. Jest ono rozwijane na zasadach *open source*. Jest alternatywą dla wspomnianych Atom.js i Visual Studio Code. Szczegóły możesz sprawdzić na oficjalnej stronie projektu: https://codelite.org/. Na rys. 4.14 przedstawiono proces instalacji programu.



Rys. 4.14. Program CodeLight (podczas instalowania tego IDE i po wybraniu opcji C/C++ kolejnym krokiem będzie zapytanie o kompilator)

4.4. Biblioteki i frameworki

Biblioteka

Dostarcza ona narzędzia do określonych działań – np. biblioteka jQuery m.in. narzędzia do prostego tworzenia animacji na stronach WWW. Biblioteka zwykle służy do rozwiązania danego problemu. Bardzo często oferuje API, dzięki któremu programista może połączyć się z funkcjonalnością zawartą w bibliotece, wybierając z niej jedynie te elementy, które są mu potrzebne. Programista rzadko dokonuje zmian w bibliotece. Możemy ją porównać do telewizora. Użytkownik ma pilot i przyciski. Dzięki klikaniu w nie otrzymuje określoną funkcjonalność. Ale czy zagląda do środka telewizora? W obecnych czasach zdarza się to niezmiernie rzadko. Przeciętny człowiek nie zastanawia się, jak działa telewizor, a jedynie korzysta z możliwości, które on oferuje. Najważniejsze jest, żeby pilot funkcjonował prawidłowo. Bardzo podobne relacje łączą programistę z biblioteką. Korzystamy z jej zasobów, zwykle nie myśląc o procesach, które zachodzą w jej wnętrzu.

CIEKAWOSTKA

Istnieje kilka sposobów dołączania bibliotek opartych na JavaScript do tworzonych projektów internetowych. Na rys. 4.15 przedstawiono jeden z nich – linkowanie wewnątrz sekcji **<head>** ... **</head>** dwóch bibliotek:

- jQuery,
- jQueryMobile.

```
<!DOCTYPE html>
<html>

<head>

    <title>Awsome Mobile App</title>
    <meta name="viewport" content="width=device-width,
    initial-scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/
    mobile/1.4.5/jquery.mobile-1.4.5.min.css" /> -
    <link rel="stylesheet" href="css/themes/darkazc.css" />
    <link rel="stylesheet" href="css/stylesheet.css" />
    <link rel="stylesheet" href="css/themes/jquery.mobile.
    icons.min.css" />

    <script src="http://code.jquery.com/jquery-1.11.1.min.js">
    </script>
    <script src="http://code.jquery.com/mobile/1.4.5/jquery.
    mobile-1.4.5.min.js"></script>

</head>

<body>
```

Rys. 4.15. Dołączenie do projektu webowego bibliotek (aplikacja mobilna) – jQuery i jQuery Mobile

Biblioteka programistyczna jest zwykle stworzona zgodnie z zasadami programowania obiektowego. Zawiera zbiór funkcji i klas służących do określonego celu. Poniżej znajdziesz kilkanaście przykładów z setek, a może nawet tysięcy dostępnych bibliotek stworzonych w najróżniejszych językach programowania.

Przykłady bibliotek:

Python

- Tkinter, strona projektu: https://wiki.python.org/moin/TkInter,
- TensorFlow, strona projektu: https://www.tensorflow.org/,
- Scikit-Learn, strona projektu: https://sklearn.org/,
- Numpy, strona projektu: https://numpy.org/,
- PyTorch, strona projektu: https://pytorch.org/,
- Eli5, strona projektu: https://eli5.readthedocs.io/en/latest/,
- SciPy, strona projektu: https://www.scipy.org/,
- Theano, strona projektu: https://pypi.org/project/Theano/,
- Pandas, strona projektu: https://pandas.pydata.org/.

CIEKAWOSTKA

Na rys. 4.16 przedstawiono dołączanie do projektu aplikacji mobilnej modułów frameworka Kivy, które dostarczą funkcjonalność pożądaną przez twórców. Dla przykładu zapis:

```
import Builder
```

umożliwi linkowanie do plików projektowych Pythona – plików języka kv, który służy do projektowania elementów interfejsu UI.

```
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.properties import
ObjectProperty
from kivy.lang import Builder
# zmiana wyglądu tła całej aplikacji
w Python
from kivy.core.window import Window
from kivy.uix.floatlayout import
FloatLayout
```

Rys. 4.16. Dołączenie bibliotek Pythona do projektu aplikacji mobilnej wykonywanej we frameworku Kivy

C++

- Xtensor, strona projektu: https://xtensor.readthedocs.io/en/latest/,
- NTL, strona projektu: https://libntl.org/,
- ALGLIB, strona projektu: https://www.alglib.net/,
- ReactPhysics3D, strona projektu: https://www.reactphysics3d.com/,
- WT, strona projektu: https://www.webtoolkit.eu/wt,
- OGRE, strona projektu: https://ogrecave.github.io/ogre/api/latest/manual.html,
- OpenGL, strona projektu: https://www.opengl.org/,
- GrphicsMagic, strona projektu: http://www.graphicsmagick.org/.

```
//importowanie bibliotek w projekcie C++
#include <iostream>
#include <conio.h>
#include <time.h>
#include <windows.h>
#include <locale.h>
```

Rys. 4.17. Dołączenie bibliotek w projekcie aplikacji C++

Framework

Jest to pewien z góry określony szkielet aplikacji (schemat), już zaprogramowany przez jego autora. Zwykle oferuje implementację rozwiązań, które często się powtarzają. Programista najczęściej pozostawia otrzymany szkielet bez zmian, a przez dodanie własnych kodów tworzy dodatkowe funkcjonalności bądź logikę biznesową (zasady działania aplikacji). Wewnątrz frameworków bardzo często są zawarte biblioteki.

Jak widać na rys. 4.18, aby zainstalować framework webowy Angular w posiadanym systemie operacyjnym (technologia: JavaScript/TypeScript), wystarczy użycie w konsoli polecenia:

`npm install -g @angular/cli` // instalacja CLI Angulara (Command-line interface).

Jeżeli wykonano powyższy krok, to instalacja każdej kolejnej, nowej instancji wymienionego frameworka odbywa się przez polecenie:

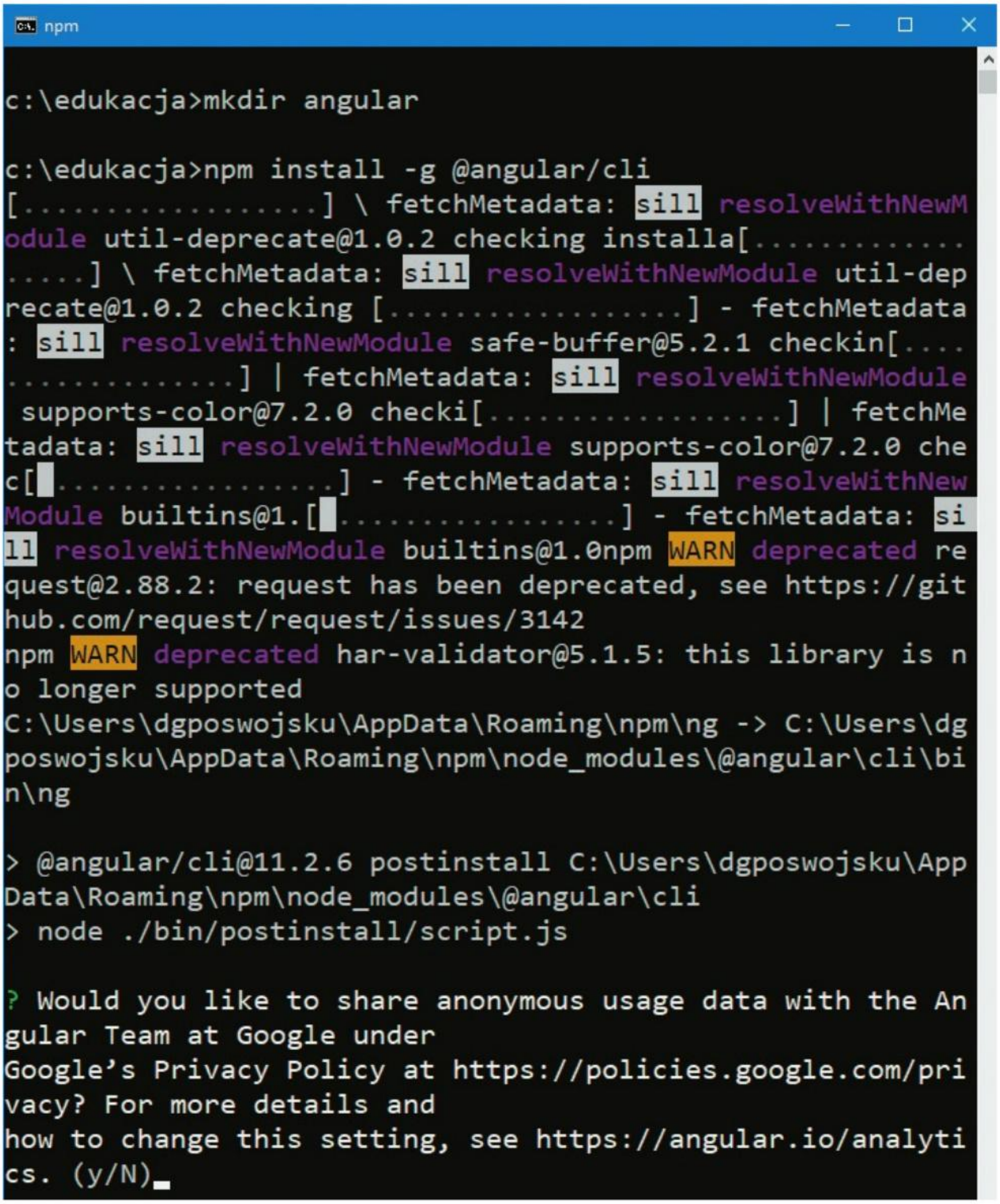
`ng new edu-app,`

gdzie edu-app to nazwa stworzonej aplikacji i katalogu, w którym instancja Angulara została stworzona.

Efektem wymienionych dwóch linijek kodu jest zainstalowany i gotowy do działania framework. Aby zainstalować Angular, trzeba pamiętać, żeby najpierw zainstalować

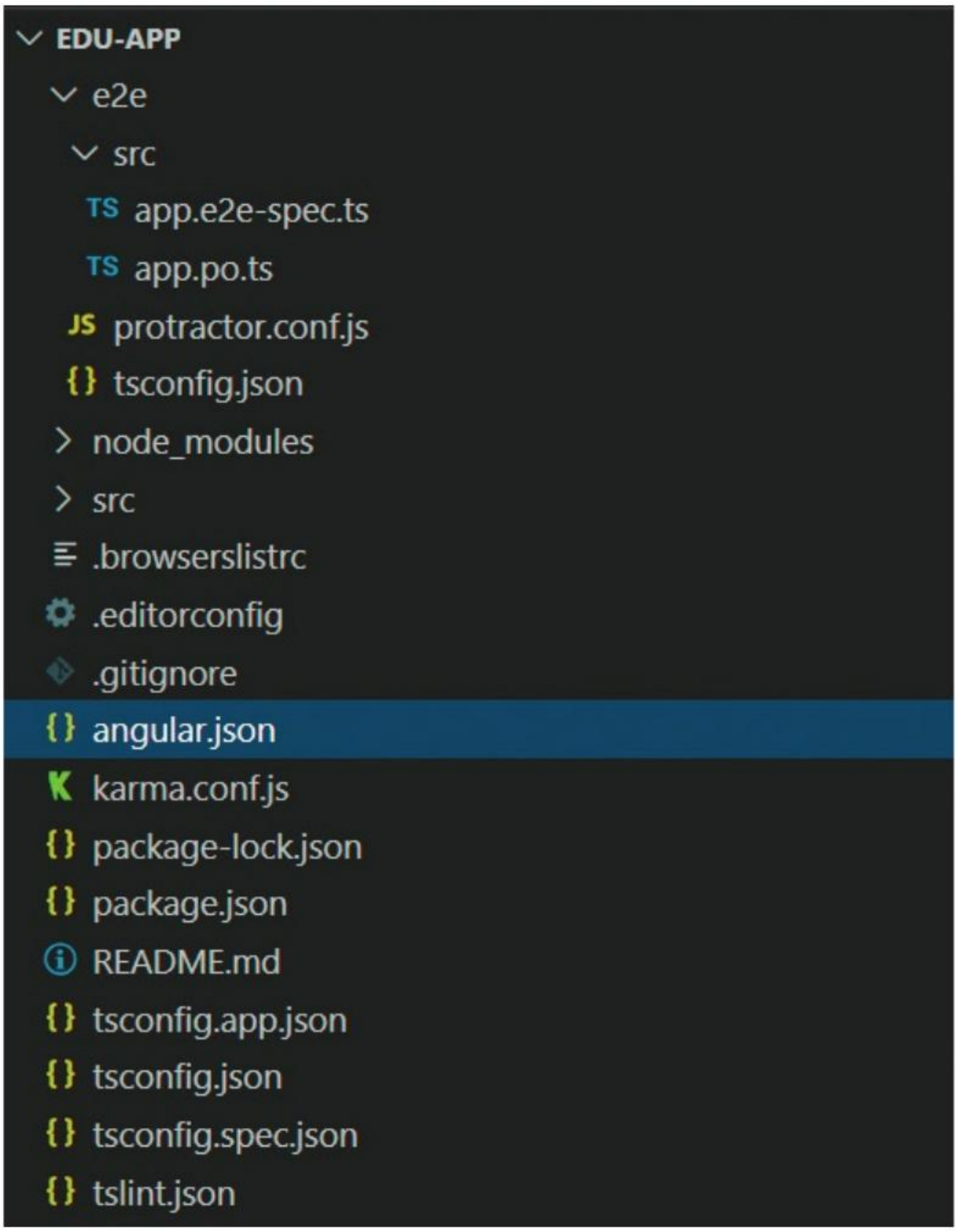
stosowne środowisko informatyczne, czyli w tym przypadku node.js (strona projektu: https://nodejs.org/en/).

Gdy framework Angular został już zainstalowany, wystarczy otworzyć folder, w którym się znajduje. Można to wykonać za pomocą dowolnego IDE (Visual Studio Code, Atom itp.). Na rys. 4.19 znajdziesz widok folderów i plików takiego zestawu oprogramowania.



Rys. 4.18. Framework Angular[3] – stworzenie katalogu i instalacja aplikacji w konsoli (środowisko node.js)

[3] Strona projektu: https://angular.io/.



Rys. 4.19. Framework Angular – widok plików i folderów po zainstalowaniu aplikacji (środowisko node.js)

Gdy zajrzysz do wnętrza pliku package.json, znajdziesz tam m.in.:

- ustawienia frameworka – zależności,
- skrypty – oferujące gotowe do działania funkcjonalności.

Jeden ze skryptów

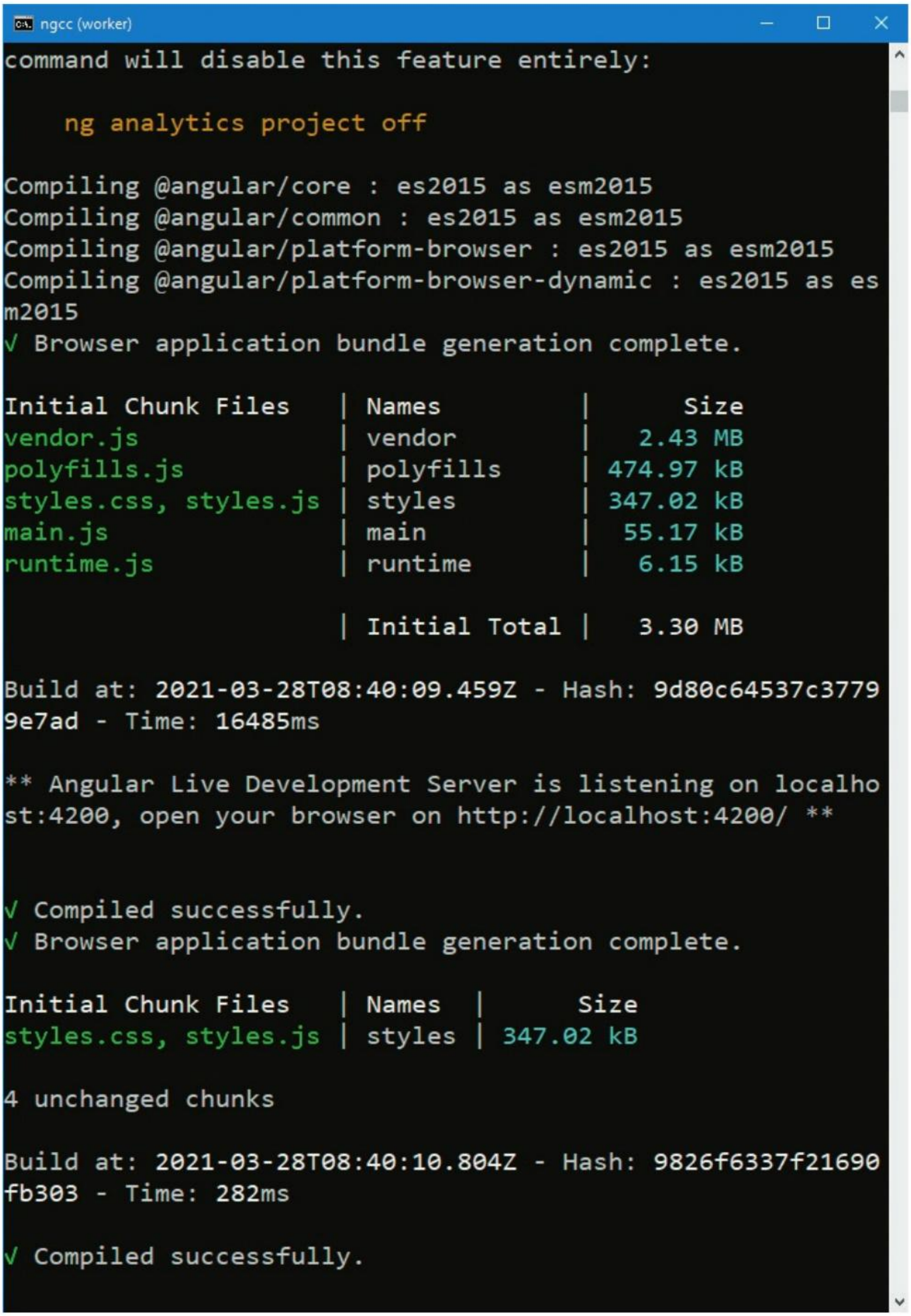
```
ng serve
```

umożliwia uruchomienie pod adresem http://localhost:4200/ podstawowej wersji frameworka Angular. Aby tego dokonać, wejdź w konsoli poleceniem:

```
cd edu-app
```

do katalogu edu-app, a następnie wykonaj polecenie:

```
ng serve.
```



Rys. 4.20. Wykonanie polecenia ng serve powoduje uruchomienie frameworka Angular – można go zobaczyć w dowolnej przeglądarce internetowej pod adresem http://localhost:4200/

Na rys. 4.20 zobaczysz widok konsoli CMD informujący o pomyślnym uruchomieniu frameworka Angular pod wspomnianym adresem.

Aby zobaczyć działający program, wystarczy przejść do przeglądarki internetowej i wpisać adres http://localhost:4200/.

Przykłady frameworków:

- **Python**
  - Kivy, strona projektu: https://kivy.org/ (jest to framework, a w nim potężne zasoby bibliotek),
  - BeeWare, strona projektu: https://beeware.org/,
  - Keras, strona projektu: https://keras.io,
  - LightGBM, strona projektu: https://lightgbm.readthedocs.io/en/latest/index.html;
- **PHP**;
  - Laravel, strona projektu: https://laravel.com/,
  - Symfony, strona projektu: https://symfony.com/;
- **C++**
  - Qt, strona projektu: https://www.qt.io/,
  - ffead-cpp, strona projektu: https://github.com/sumeetchhetri/ffead-cpp.

Warto podkreślić, że czasem podział na bibliotekę i framework jest dość trudny do wykonania, a wiele osób z branży nawet nie widzi zbyt wielkiej różnicy między biblioteką a frameworkiem. Co ważne, jest ona dość duża.

Dla przykładu weźmy Kivy – pythonowy, właśnie, framework czy bibliotekę? Nawet na stronie głównej można przeczytać, że jest to (źródło: https://kivy.org/#home): „Biblioteka Pythona *open source* do szybkiego rozwoju aplikacji", a kilka linijek niżej: „Framework jest stabilny i ma dobrze udokumentowany interfejs API, a także przewodnik programowania [...]".

Kivy będziemy używali w jednym z kolejnych rozdziałów i wówczas będzie można samodzielnie określić, czy jest on frameworkiem, czy biblioteką.

4.5. Kompilowanie i uruchamianie programów

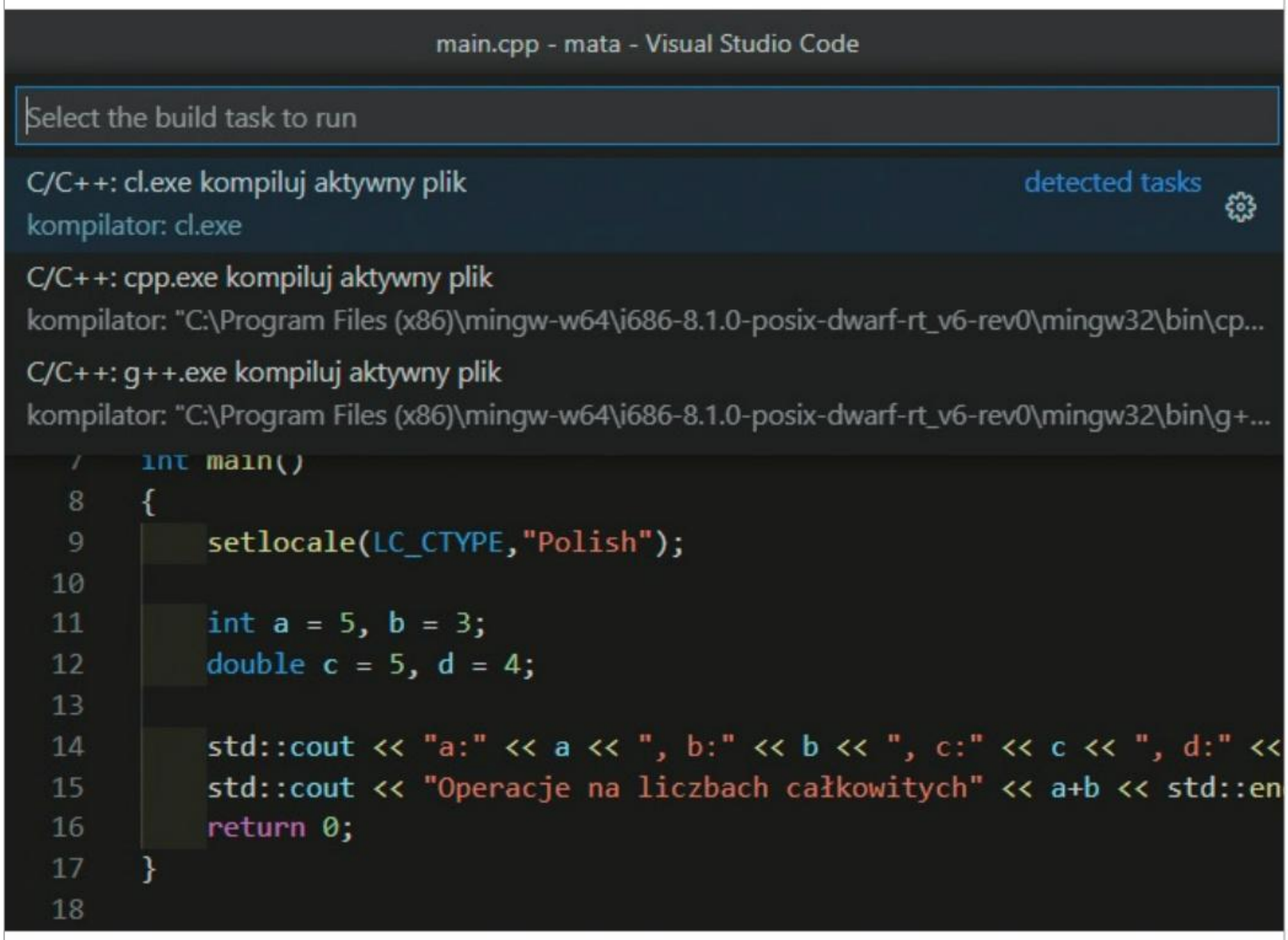
Proces kompilacji jest bardzo skomplikowany. Na szczęście obsługuje go wiele wydajnych programów nazywanych **kompilatorami** (właściwymi dla danego języka). W internecie możesz znaleźć ich wiele do poszczególnych języków wymagających tego procesu.

Na rys. 4.21 znajdziesz widok programu Visual Studio Code. Gdy zainstalujesz odpowiednie dodatki, możesz w nim przeprowadzać procesy debugowania i kompilacji plików języka programowania C++.

CIEKAWOSTKA

Wejdź na ciekawą stronę dotyczącą C++: https://stroustrup.com/index.html. Należy ona do twórcy języka C++ i choćby ze względu na jego zasługi w procesie tworzenia tej potężnej technologii koniecznie musisz poznać wspomniany portal. Odszukasz tam mnóstwo materiałów edukacyjnych przydatnych do nauki. Na podstronie https://stroustrup.com/compilers.html znajduje się lista kompilatorów do C++ (zarówno płatnych, jak i bezpłatnych). Oto kilka najbardziej znanych:

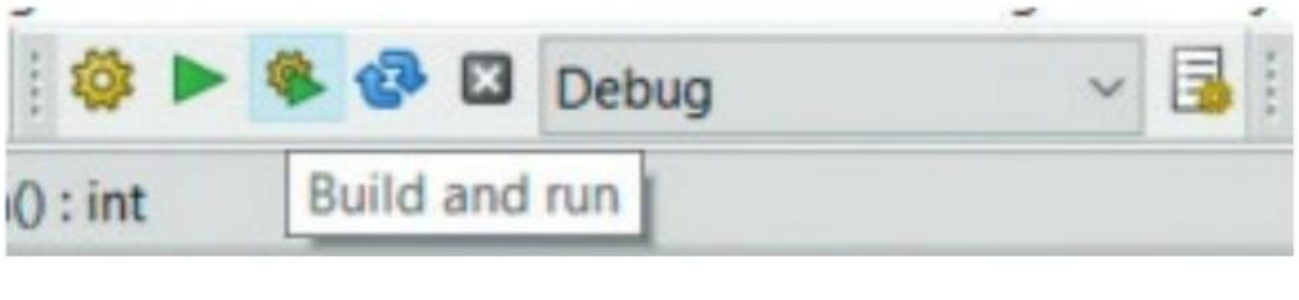
- MINGW – https://sourceforge.net/projects/mingw/files/,
- GNU CC – https://gcc.gnu.org/,
- Bloodshed Dev-C++ – http://www.bloodshed.net/,
- Clang C++ – https://clang.llvm.org/,
- Cygwin (GNU C++) – https://www.cygwin.com/,
- Microsoft C++ – https://visualstudio.microsoft.com/pl/downloads/,
- Apple C++ Xcode – https://developer.apple.com/xcode/.



Rys. 4.21. Visual Studio Code – widok wyboru ustawień kompilatora w procesie uruchamiania zadań

CIEKAWOSTKA

Program CodeBlocks jest bardzo często używany w szkołach. Ma funkcje debugowania, kompilowania i uruchamiania plików języka programowania C++. Jego niewątpliwą zaletą jest to, że wszystkie wymienione funkcjonalności są dostępne w standardowej instalacji. To jest zapewne jeden z powodów jego popularności – bez dodatkowego wysiłku można prowadzić na nim lekcje programowania w C++.



Rys. 4.22. CodeBlocks – widok paska narzędziowego Build and run

SPRAWDŹ SWOJĄ WIEDZĘ

1. Opisz różnice między pojęciami:
   - kompilator,
   - interpreter,
   - debugger.
2. Wymień etapy kompilacji kodu.
3. Czym się różni kompilacja od interpretacji kodu? Wypisz podstawowe zalety i wady każdego z rozwiązań.
4. Opisz zagadnienia „biblioteka” i „framework”. Porównaj je ze sobą. Wskaż przykłady bibliotek z co najmniej trzech języków programowania.
5. Scharakteryzuj pojęcie kompilowania programów.
6. Co to jest uruchamianie programów?
7. Opisz przykładowe zestawy środowisk do tworzenia i uruchamiania aplikacji desktopowych dla języków programowania:
   - C++,
   - C#,
   - Python,
   - JAVA,
   - JavaScript.
8. Opisz przykładowe zestawy środowisk do tworzenia i uruchamiania aplikacji mobilnych dla języków programowania:
   - Python,
   - C++,
   - C#,
   - Java,
   - JavaScript.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Aby wykonać zadanie, potrzebujesz programu CodeBlocks. Możesz go pobrać z oficjalnej strony projektu https://www.codeblocks.org/ i zainstalować w swoim systemie operacyjnym.
   - W programie CodeBlocks przygotuj prosty projekt konsolowy, np. dokonujący wyliczeń matematycznych 2 + 3 i pokazujący wynik w konsoli.
   - Poszukaj w menu CodeBlocks zakładki Settings->Complier i ją otwórz. W nowej otwartej karcie znajdziesz od góry – rozwijany element menu Selected compiler. Uruchom napisany przez siebie program co najmniej pięć razy. Za każdym razem wybierz w sekcji Selected compiler inny kompilator.
   - Zapisz w edytorze tekstu swoje spostrzeżenia. Wskaż użyty kompilator i napisz, co się wydarzyło po jego użyciu.
2.
   - W systemie operacyjnym używanego komputera zainstaluj program nodejs – wersję instalacyjną znajdziesz na stronie https://nodejs.org/en/.
   - Zgodnie z instrukcjami zawartymi w tym rozdziale zainstaluj framework Angular – najpierw w systemie, a potem jego instancję w wybranym katalogu.
   - Poleceniem ng serve aktywuj wskazany framework.
   - Uruchom framework Angular w dowolnej przeglądarce internetowej pod adresem http://localhost:4200/.
   - Poświęć co najmniej 15 minut na sprawdzenie funkcjonalności stworzonej aplikacji. Znajdziesz w niej m.in. informacje pomocne do rozwijania projektu.

5 Podstawowe zasady programowania

ZAGADNIENIA

- Funkcje – kody do wielokrotnego stosowania
- Rekurencja
- Programowanie jako zapis algorytmów

5.1. Funkcje – kody do wielokrotnego stosowania

Od tego działu rozpoczynamy pracę z językami programowania, w szczególności zajmiemy się Pythonem (choć wspomnimy także o kilku innych, m.in. C++ i JavaScript). Python to jeden z najszybciej rozwijających się projektów informatycznych. Warto podkreślić jego sposób licencjonowania – jest to *open source*, choć można w nim tworzyć także komercyjne aplikacje w celu sprzedaży zarobkowej.

Python ma zastosowanie w wielu dziedzinach naszego życia. Szczególnie istotne jest jego wykorzystanie w zakresie sztucznej inteligencji. My oczywiście aż tak zaawansowanych projektów w tym podręczniku nie będziemy robili, ale i tak potrzebna będzie choćby podstawowa znajomość tego wspaniałego i potężnego języka. Warto zajrzeć do przepastnych zasobów internetowych i poznać choćby podstawy Pythona, np. na jego stronę domową https://www.python.org/ i podstronę, zawierająca dokumentację do najnowszej wersji Pythona i jego starszych odpowiedników – https://docs.python.org/3/.

Jeżeli nie masz jeszcze zainstalowanego Pythona na swoim komputerze, zrób to teraz. Na rys. 5.1 znajdziesz pokaz instalacji w systemie Linux. Do wykonania ćwiczeń zawartych w dalszej części tego podręcznika niezbędny jest Python z linii 3.x, czyli Python 3.x. Aby pracować na frameworku Kivy opisywanym w czwartym rozdziale, nie powinna to być wersja niższa niż 3.6, a najlepiej 3.8 bądź 3.9. Jest to bardzo ważne, szczególnie gdyby się okazało, że masz zainstalowany Python 2.x. Jest to stara wersja tego języka programowania, której wsparcie już właściwie się skończyło, ale na niektórych Linuksach może być wciąż spotykana. Trzeba sprawdzić, którą wersję masz w systemie. Możesz to zrobić poleceniem:

- python --version
- python2 --version
- python3 --version

```
dg@DESKTOP-3NA7D8I:~$ python --version
dg@DESKTOP-3NA7D8I:~$ python2 --version

Command 'python2' not found, but can be
installed with:

sudo apt install python2

dg@DESKTOP-3NA7D8I:~$ python3 --version
Python 3.8.5
dg@DESKTOP-3NA7D8I:~$
```

Rys. 5.1. Sprawdzenie zainstalowanych wersji Python w systemie operacyjnym Linux – terminal

Spójrz na rys. 5.1 i 5.2. Zauważ, że nieco odmiennie na polecenia reagują Linux i Windows – temu pierwszemu trzeba sprecyzować, o którą wersję chcemy się zapytać. W drugim z wymienionych systemów wystarczy wpisać zapytanie o wersję Pythona, bez wyszczególniania, o którą linię nam chodzi. Co ciekawe, w systemie operacyjnym może być zainstalowanych kilka wersji Pythona, w tym jedna defaultowa, czyli podstawowa.

Jeżeli dysponujesz systemem Linux, to zajrzyj na stronę https://www.python.org/downloads/source/. Znajdziesz na niej pliki instalacyjne i informacje niezbędne do zainstalowania Pythona. W przypadku Windows 10 możesz pobrać środowisko najnowszego Pythona i zainstalować bezpośrednio ze sklepu Windows Store. Tak (przynajmniej na początku) będzie najłatwiej. W obu systemach poprawność instalacji możesz sprawdzić powyżej wypisanymi poleceniami. Jeżeli Python się nie zgłasza, to w przypadku Windowsa zapewne nie został podczas instalacji dodany do ścieżki PATH w Zmiennych Środowiskowych systemu. Wtedy trzeba dodać go ręcznie, np. wpisy w PATH wyglądają tak:

```
C:\Program Files\Python39\Scripts\

C:\Program Files\Python39\.
```

```
C:\>python --version
Python 3.9.2

C:\>python2 --version
'python2' is not recognized as an internal
or external command,
operable program or batch file.

C:\>python3 --version
Python 3.9.2

C:\>
```

Rys. 5.2. Sprawdzenie zainstalowanych wersji Pythona w systemie operacyjnym Windows – konsola CMD



Rys. 5.3. Instalacja języka Python 3.9 z opcją dodania do PATH (Zmienne Środowiskowe)

Pracując nad rozwojem różnego rodzaju aplikacji, programiści tworzą bardzo dużo linijek kodu. Zapewne musieliby pisać co najmniej kilka tysięcy razy więcej, gdyby nie powstały specjalne funkcje. Są one odpowiedzią na:

- złożoność kodu – dzięki funkcji widoczny kod jest prostszy do zrozumienia;
- powtarzalność kodu – zamknięcie kodu w funkcji powoduje, że możemy go użyć wielokrotnie;
- zużycie czasu – gdy ten sam kod zawarty w funkcji możesz użyć tyle razy, ile chcesz, wykonując jedynie wywołanie funkcji (rys. 5.4).

funkcjaProsta.py ×

funkcjaProsta.py > rozwiazanie

```
1   rozwiazanie = 0
2   szukanaLiczba = rozwiazanie + 1
3   if szukanaLiczba < 3:
4       szukanaLiczba = 10 + szukanaLiczba
5       rozwiazanie = szukanaLiczba * 3
6   else:
7       rozwiazanie = 0
8
9   print(rozwiazanie)
10
11
12  def szukana(rozwiazanie2):
13      szukanaLiczba = rozwiazanie2 + 1
14      if szukanaLiczba < 3:
15          szukanaLiczba = 10 + szukanaLiczba
16          rozwiazanie2 = szukanaLiczba * 3
17      else:
18          rozwiazanie2 = 0
19      return rozwiazanie2
20
21
22  print(szukana(0))
23
```

Rys. 5.4. Python – kod programistyczny pokazujący kodowanie bez użycia funkcji i z użyciem tego elementu

Na rys. 5.4 znajduje się kod Pythona:

a) linijki 1–7 – bez wykorzystania funkcji,

b) linijki 12–19 – z wykorzystaniem funkcji.

Kodu wskazanego w punkcie A nie możesz nigdzie ponownie użyć, natomiast kod będący funkcją (punkt B) można wykorzystać w innej części budowanej aplikacji przez wywołanie funkcji, czyli tutaj:

```
szukana()
```

W języku programowania Python:

- def – oznacza definicję funkcji;
- wywołanie / wypisanie nazwaFunkcji() – oznacza wywołanie funkcji.

Więcej przykładów tworzenia i zastosowania funkcji znajdziesz w kolejnych rozdziałach podręcznika. Spotkasz się także z pojęciem **„metoda"**. Czym różni się ona od funkcji? Są to bardzo zbliżone pojęcia, niektórzy mówią, że to dokładnie to samo. Mają trochę racji, gdyby rozpatrywać obszar inny niż programowanie obiektowe. Natomiast w tym sposobie pisania kodu metoda jest przypisana do obiektu. Zarówno metoda, jak i funkcja odpowiadają za określone działanie, a raczej za dynamikę kodu. Kolokwialnie rzecz ujmując, funkcję (oraz metodę) można przyrównać w języku polskim do czasownika. Przy czym jeżeli konstrukcja taka jak wskazana na rys. 5.4 (def) występuje:

- w obiekcie – mamy do czynienia z nazwą metoda;
- poza obiektem – nazywamy ją funkcją.

obiektTest.py

```
1
2   class TestObiekt:
3
4       zmienna = "Jestem polem klasy,
        czyli właściwością"
5
6       def drukuj(self):
7           return ("Jestem metodą klasy")
8
9   nowyObiekt = TestObiekt()
10
11  print(nowyObiekt.drukuj())
12  print(nowyObiekt.zmienna)
13
```

Rys. 5.5. Kod programistyczny Python pokazujący kod prostego obiektu i dostęp do metody i właściwości (pola) obiektu

5.2. Rekurencja

Rekurencja polega na umieszczeniu w treści funkcji (lub szerzej – procedury) odwołania do tej właśnie funkcji. Możemy nawet użyć kolokwializmu: „funkcja wywołuje samą siebie, aż osiągnie określone założenie (lub cele)".

Sprawdźmy powyższą definicję na konkretnych przykładach.

```
def policzSilnia(param):
    if param < 1:
        return 1
    else :
        return param * policzSilnia(param - 1)

print(policzSilnia(4))
```

Rys. 5.6. Kod programistyczny Python pokazujący wykorzystanie rekurencji do wyliczenia pojęcia matematycznego – silnia

Alternatywą dla rekurencji jest zastosowanie **metody iteracyjnej**, czyli kolejnych powtórzeń w pętli (np. **`for`**).

W innych językach programowania rekurencja jest również bardzo często stosowana. Przykładowo: w JavaScript animacje są oparte właśnie na tym elemencie wiedzy programistycznej – wykorzystują wbudowaną metodę **`requestAnimationFrame()`**. Przykładowy kod znajdziesz na rys. 5.7.

Zgodnie z tym, co widać na rys. 5.7, metoda **`requestAnimationFrame()`**, zawarta i wykonana wewnątrz funkcji **`ruszDivRequest()`**, wywołuje funkcję, w której wnętrzu (tzw. ciało funkcji) się znajduje. Dzięki temu obiekt (tutaj: kulka) przesuwa się w każdym ruchu o 2 px. Mówi nam o tym zmienna **`ruchRequest`**.

W technologiach internetowych opartych na zestawie HTML, CSS, JavaScript istnieją dwie podstawowe możliwości wykonania animacji elementów strony internetowej – z wykorzystaniem:

- CSS, w tym jednej z bardzo wielu bibliotek opartych na tej technologii, np.: Animate.css (strona projektu: https://animate.style/), Motion UI (strona projektu: https://github.com/foundation/motion-ui), AnimXYZ (strona projektu: https://animxyz.com/), CSShake (strona projektu: http://elrumordelaluz.github.io/csshake/#1);
- JavaScript, w tym jednej z bardzo wielu bibliotek opartych na tym języku programowania, np.: jQuery (strona projektu: https://jquery.com/), Bounce.js (strona projektu: http://bouncejs.com/), jQuery (strona projektu: https://jquery.com/), Anime.js (strona projektu: https://animejs.com/).

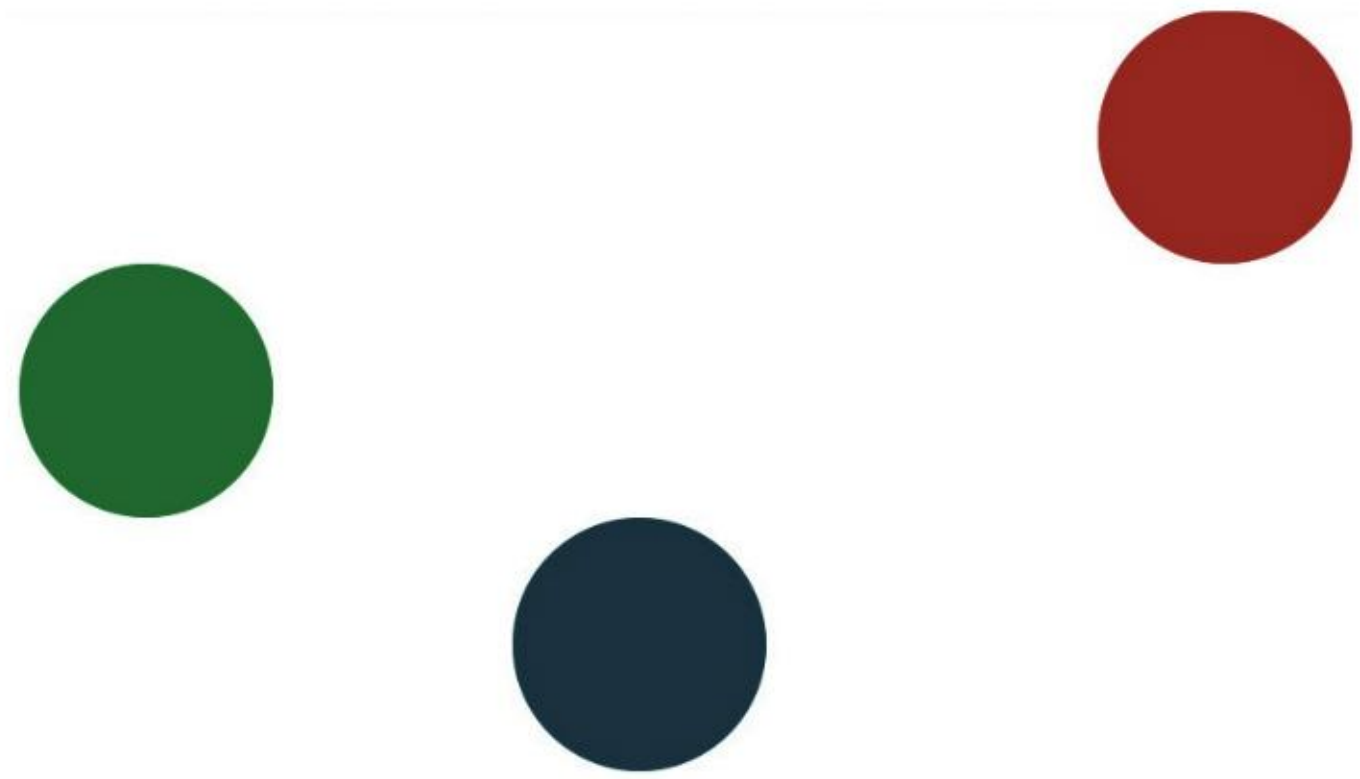
```
// Animacja kulki z wykorzystaniem metody
requestAnimationFrame
var animowanyDivSetIntervalRequest = document.
getElementById('divRequestAnimationFrame');
animowanyDivSetIntervalRequest.style.position =
'relative';
var pozycjaDivRequest = 0;

function ruszDivRequest() {

    if (pozycjaDivRequest < szerokoscOknaAnimacji - 350)
    {
        var ruchRequest = 2;
        pozycjaDivRequest = pozycjaDivRequest +
        ruchRequest;
        animowanyDivSetIntervalRequest.style.left =
        pozycjaDivRequest + 'px';

    }
    requestAnimationFrame(ruszDivRequest);
}
ruszDivRequest();
```

Rys. 5.7. JavaScript – kod programistyczny pokazujący rekurencję



Rys. 5.8. JavaScript – widok na stronie internetowej animowanych kulek

5.3. Programowanie jako zapis algorytmów

Python jest językiem programowania, którego jednym z głównych celów jest opisanie, co ma być zrobione (programowanie imperatywne). Oczywiście tego typu języków programowania jest dużo więcej, np. Perl, PHP, Java czy JavaScipt.

Alternatywą są języki działające na zasadach programowania deklaratywnego. Wówczas programista opisuje, co chcemy osiągnąć, czyli wskazuje warunki, jakie musi spełniać końcowe rozwiązanie. Nie narzuca on zaś kodem sekwencji kroków, które prowadzą do pożądanego rozwiązania. Przykładowymi językami deklaratywnymi są SQL, HTML i CSS.

Python jest językiem, w którym programujemy imperatywnie, a zapisany kod jest bardzo czytelny (wcięcia ustawiają bloki kodu), więc idealnie nadaje się do przekładania algorytmów na kod komputerowy.

W takim razie napiszmy w Pythonie pierwszy kod, który można przedstawić tak, jak na rys. 5.9.



Rys. 5.9. Przykładowy algorytm

Na rys. 5.10 i 5.11 przedstawiono pythonowy kod, który przekłada algorytm zapisany w formie graficznej na działający program. W komentarzach opisano poszczególne linijki kodu.

algorytm.py

```
1   liczbaStartowa = 4
2   daneInput = int(input('Wpisz swoją
    ulubioną liczbę, jeżeli to będzie taka
    sama liczba jak moja, będę mógł złożyć
    Tobie gratulacje, a jeżeli nie będzie
    - no cóż: nie zawsze sie wygrywa! '))
3
4   # To jest poczatek instrukcji
5   if daneInput == liczbaStartowa:
6       # To jest początek tzw. ciała
        instrukcji if
7       print('Jesteś super, prawie jak
        prorok, skoro zgadłeś liczbę,
        którą wymyśliłem. GRATULUJĘ!')
8       # A to jest koniec pierwszego
        bloku (tzw. ciała) instrukcji if
```

Rys. 5.10. Algorytm wyboru zaimplementowany w języku programowania Python – część 1

UWAGA

Dane przekazywane z klawiatury do danego języka programowania, np. Pythona – bez względu na to, czy użytkownik będzie wpisywał w konsoli bądź w stworzonym polu **`input`** – będą typu **`string`**. Nawet jeżeli zostanie wprowadzona liczba, np. 55, to i tak dla komputera będzie to zwykły string, czyli „55”. Dla świadomego tego faktu programisty nie będzie to stanowiło problemu, gdyż wystarczy użyć właściwego elementu języka programowania, który spróbuje zamienić string na liczbę. Przykładem w języku programowania Python może być zapis na rys. 5.10:

daneInput = int(input("Wpisz swoją ulubioną liczbę, jeżeli to będzie taka sama liczba jak moja, będę mógł złożyć Tobie gratulacje, a jeżeli nie będzie – no cóż: nie zawsze się wygrywa! ")).

Korzystając z konstrukcji **int()**, podany znak będzie zamieniony na liczbę, oczywiście o ile wprowadzi ją użytkownik z klawiatury. Jeżeli jednak wprowadzi inny znak, silnik Pythona pokaże błąd, np.:

"ValueError: invalid literal for int() with base 10: 'sdf'".

```
    # Zaczynamy drugi blok
    elif daneInput < liczbaStartowa:
        # To jest początek drugiego tzw.
        ciała instrukcji if, wykona się
        ta część kodu, gdy nie wykona się
        instrukcja po if lub jak wolisz,
        gdy wpisana liczba będzie mniejsza
        od liczby wskazanej na początku
        tego prostego algorytmu
        print('Niestety, podałeś zbyt
        małą liczbę :(')
        # A to jest koniec drugiego bloku
        (tzw. ciała) instrukcji if
    # Zaczynamy trzeci blok
    else:
        # To jest początek trzeciego
        bloku, który wykona się jedynie
        wtedy, gdy nie wykona się żaden z
        pozostałych
        print('Niestety, podałeś zbyt
        dużą liczbę :(')
        # A to jest koniec trzeciego bloku
        (tzw. ciała) instrukcji if
```

Rys. 5.11. Algorytm wyboru zaimplementowany w języku programowania Python – część 2

SPRAWDŹ SWOJĄ WIEDZĘ

1. Omów zagadnienie instalacji języka Python w Windows i w Linux. Dlaczego w Windows w Zmiennych Środowiskowych muszą się znaleźć w ścieżce PATH odpowiednie wpisy związane z Pythonem?
2. Co to jest funkcja w programowaniu? Do czego jest wykorzystywana ta konstrukcja programistyczna?
3. Opisz zasady stosowania rekurencji w wybranym języku programowania.
4. Czy Python jest dobrym językiem do tworzenia algorytmów? Uzasadnij odpowiedź.
5. Stwórz w edytorze tekstu przykładowy algorytm w sposób opisowy, a potem zaprogramuj go w Pythonie lub innym języku programowania.
6. Stwórz algorytm, który będzie wykorzystywał rekurencję.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz w języku programowania Python 3.x program wyświetlający twoje dane w konsoli. W tym celu:
   A. stwórz nowy plik o nazwie mojeDane.py;
   B. utwórz w pliku, o którym mowa w punkcie A, trzy zmienne o dowolnych nazwach;
   C. do zmiennych stworzonych w punkcie B przypisz swoje dane – odpowiednio: imię, nazwisko i datę urodzenia;
   D. w konsoli uruchom program przygotowany w punktach A, B i C – wpisz polecenie python mojeDane.py.

   Pamiętaj, że wykonując polecenie z punktu D, musisz znajdować się w tej samej lokalizacji co plik mojeDane.py.

6 Typy danych w programowaniu na bazie projektu Gra edukacyjna Zgadnij Liczbę

ZAGADNIENIA

- Proste i złożone typy danych
- Projekt Gra Edukacyjna Zgadnij Liczbę – deklarowanie zmiennych różnych typów danych
- Projekt Gra Edukacyjna Zgadnij Liczbę – operacje na zmiennych wejścia i wyjścia
- Projekt Gra Edukacyjna Zgadnij Liczbę – typy złożone i operacje na nich na bazie listy
- Deklarowanie własnych typów danych

6.1. Proste i złożone typy danych

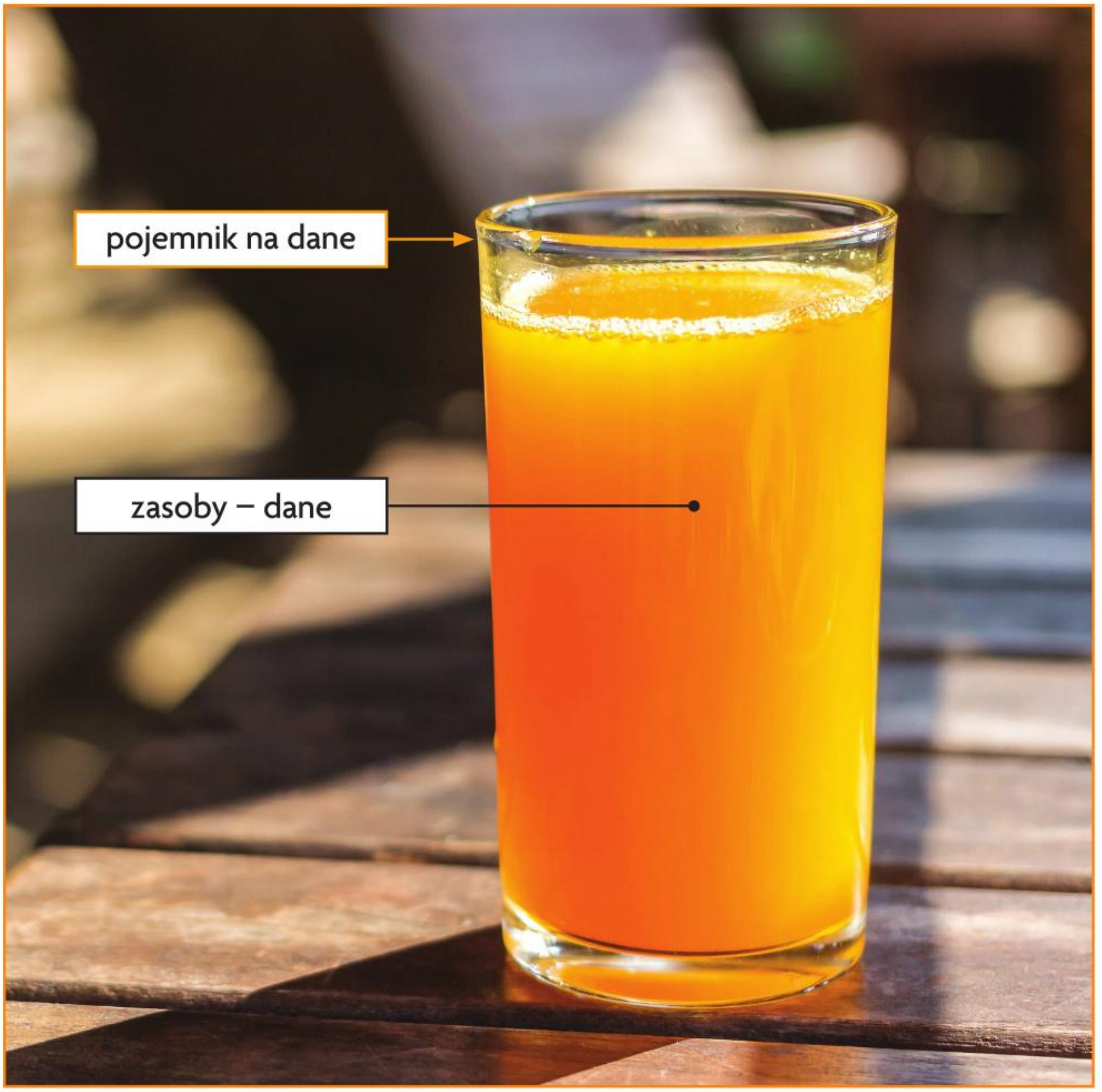
Na rys. 6.1 umieszczono krótkie przypomnienie wiedzy o podstawowych typach danych, opierając się na języku C++.

```
/** PODSTAWOWE TYPY, JAKIE MOGĄ PRZYJĄĆ NOWE
ZMIENNE */
/// Każdy typ danych zajmuje właściwą sobie ilość
miejsca w pamięci RAM
// liczba całkowita
int a = 1;
// liczba zmiennoprzecinkowa (wymierna) z małego
zakresu
float b = 1.5;
// liczba zmiennoprzecinkowa (wymierna) z dużego
zakresu
double c = 99.54782911;
// znak LUB liczba z zakresu od 0 do 255
char d = 'X' ;
// napis - ciąg znaków
string e = "to jest napis";
```

Rys. 6.1. Przykłady prostych typów danych w języku C++ – typy: liczbowe, znak i łańcuch znaków

Typ danych określa rodzaj pojemnika (lub odwrotnie), do którego możemy włożyć i w nim przetrzymywać określone zasoby. W praktyce informatycy dzielą typy danych według różnych kryteriów. Jednym z najważniejszych jest podział na typy:

- proste,
- złożone.



Rys. 6.2. Właściwie dobrany pojemnik na dane określonej wielkości

Typy proste danych, często określane jako prymitywne, przechowują tylko „surowe" dane, czyli takie jak:

- liczby stałe (np. 10, 200),
- znaki,
- litery (zarówno z tradycyjnego alfabetu łacińskiego, jak i inne),
- wyrazy – ciągi znaków.

```
// Deklaracja tablicy jednowymiarowej,
zawierającej 4 liczby całkowite.
int[] tablicaJednowymiarowa = new int[4];

// Przypisanie do indeksu 1 (czyli do drugiej
pozycji w tablicy, gdyż indeksujemy od 0)
liczby 44
tablicaJednowymiarowa[1] = 44;

//Deklarowanie i ustawianie wartości elementów
tablicy indeksowanej numerycznie o typie
całkowitoliczbowym.
int[] tablica2 = new int[] { 0, 3, 51, 9, 22 };

//Deklarowanie i ustawianie wartości elementów
tablicy indeksowanej numerycznie o typie
łańcuchowym (ciąg liter/znaków).
string[] tablicaString = new string[] {
"Witaj", "w tablicy", "zawierającej", "litery"
};
```

Rys. 6.3. Przykłady złożonych typów danych, język C# – tablice indeksowane numerycznie o typach całkowitoliczbowym i string (łańcuch znaków)

Typy złożone danych (przykład na rys. 6.3) mają różne nazwy w poszczególnych językach programowania i mogą oferować wiele wariantów. Jako przykłady z różnych języków programowania można wymienić typy:

- tablice indeksowane numerycznie,
- tablice asocjacyjne,
- rekordowe:
  - struct,
  - unia,
- kolekcje:
  - wskaźniki,
  - wektory,
  - listy,
  - stosy,
  - kolejki,
- plikowe.

W dalszej części tego podręcznika stworzymy kody, które będą operowały na różnych typach danych. Aby pójść dalej, musisz przypomnieć sobie wiedzę zarówno z zakresu typów danych złożonych, jak i prostych, a także zrozumieć zasadę związaną bezpośrednio z pojęciami programistycznymi:

- **typowanie statyczne** (np. w C++, C#, Java) – nadawanie typów zmiennym w czasie kompilacji programu (pojemnik na dane określa typ danych, a dane trzeba dopasować do pojemnika);
- **typowanie dynamiczne** (np. w Python, JavaScript, PHP) – typy są przypisywane do wartości (typ danej przypisanej do pojemnika ustala, jakiego rodzaju typem danych będzie ten pojemnik).

6.2. Projekt Gra edukacyjna Zgadnij Liczbę – deklarowanie zmiennych różnych typów danych

Wykonamy projekt Gra edukacyjna Zgadnij Liczbę. Krok po kroku napiszemy kod w języku programowania Python. Warto przypomnieć zasadę pisania kodu Pythona: kolejne bloki są tworzone za pomocą wcięć poszczególnych linii. Każde wcięcie powinno być iloczynem najmniejszego. Przykładowo: jeżeli pierwsze ma szerokość tabulacji, to drugi poziom musi mieć szerokość wcięcia o rozmiarze dwóch tabulacji, trzeci – trzech itd. Zaraz zobaczysz to w kodzie działającego programu.

Zaczniemy od przemyślenia (opisania) projektu, potem zadeklarujemy zmienne, stworzymy logikę, a na końcu dodamy element losowości, wykorzystując jedną z bibliotek Pythona.

Gra edukacyjna Zgadnij Liczbę – opis projektu (pseudokod)

Z zakresu liczb od 0 do 99 (obie wartości skrajne mogą zostać wylosowane) losujemy jedną liczbę całkowitą, którą użytkownik będzie musiał odgadnąć.



Rys. 6.4. Gra edukacyjna Zgadnij Liczbę

Użytkownik będzie wpisywał po jednej liczbie (w konsoli). Po każdym wpisaniu uzyska jedną z odpowiedzi:

A. Szukana liczba jest większa, spróbuj ponownie wpisać jakąś liczbę – gdy użytkownik poda liczbę za małą.

Zaistnienie przypadku A oznacza kontynuowanie gry, czyli pojawienie się kolejnego pola umożliwiającego wpisanie kolejnej liczby.

B. Szukana liczba jest mniejsza, spróbuj ponownie wpisać jakąś liczbę – gdy użytkownik poda liczbę za dużą.

Zaistnienie przypadku B oznacza kontynuowanie gry, czyli pojawienie się kolejnego pola umożliwiającego wpisanie kolejnej liczby.

C. Brawo, gratuluję, zgadłeś liczbę! To faktycznie jest liczba. Od nowej linijki powinna być wypisana szukana i znaleziona liczba, gdy poda szukaną liczbę.

Zaistnienie przypadku C kończy grę.

Po zakończeniu gry muszą pojawić się informacje dotyczące:

- zakończenia gry;
- liczby ruchów (zsumowanych, pojedynczych zgadnięć);
- wszystkich liczb, które użytkownik wpisał podczas zgadywania.

Gra edukacyjna Zgadnij Liczbę – kodowanie

Na początek stwórz plik zgadnijLiczbe.py w katalogu przeznaczonym do nauki. Kolejny krok to losowe wybranie cyfry przypisanej do zmiennej. Losowaniem zajmiemy się później, ponieważ będzie to wymagało zaimportowania stosownej biblioteki. Na razie wybierzemy liczbę – np. 55:

```
liczbaStartowa = 55
```

W trakcie działania programu będą pojawiały się informacje dla użytkownika. Aby nie musieć przepisywać tekstów, przypiszemy je do zmiennych. Za każdym razem, gdy będziemy chcieli poinformować użytkownika o stanie gry, skorzystamy wówczas z jednej ze stworzonych zmiennych:

```
LiczbaJestWieksza = "Szukana liczba jest większa, spróbuj
ponownie wpisać jakąś liczbę"

LiczbaJestMniejsza = "Szukana liczba jest mniejsza, spróbuj
ponownie wpisać jakąś liczbę"

ZgadzaSie = "Brawo, gratuluję, zgadłeś liczbę! To faktycznie
jest liczba: "

liczbaRuchowInfo = "Liczba Twoich ruchów potrzebnych do
zgadnięcia właściwej liczby to: "

liczbaRuchow = 0

podawaneLiczby = "podczasZgadywania podałeś liczby: "
```

W ramach projektu przewidzimy także tablicę indeksowaną numerycznie, która w języku Python nazywa się list, czyli po polsku – lista.

```
tablicaPodawanychLiczba = []
```

Na rys. 6.5 przedstawiono kod w zakresie stworzonych zmiennych w naszym projekcie.

```
liczbaStartowa = 55

aLiczbaJestWieksza = "Szukana liczba jest większa,
spróbuj ponownie wpisać jakąś liczbę"
bLiczbaJestMniejsza = "Szukana liczba jest
mniejsza, spróbuj ponownie wpisać jakąś liczbę"
cZgadzaSie = "Brawo, gratuluję, zgadłeś liczbę!
To faktycznie jest liczba: "
liczbaRuchowInfo = "Liczba Twoich ruchów
potrzebnych do zgadnięcia właściwej liczby to: "

liczbaRuchow = 0

podawaneLiczby = "Podczas zgadywania podałeś
liczby: "

tablicaPodawanychLiczb = []
```

Rys. 6.5. Projekt Gra edukacyjna Zgadnij Liczbę – deklarowanie zmiennych różnych typów danych

6.3. Projekt Gra edukacyjna Zgadnij Liczbę – operacje na zmiennych wejścia i wyjścia

Po uruchomieniu gry powinny ukazać się: przywitanie użytkownika i informacja o potrzebie wpisania liczby.

```
daneInput = int(input("Witaj serdecznie. Aby
rozpocząć grę, wpisz liczbę z przedziału od 1 do
99, która Twoim zdaniem została wylosowana : "))
```

Rys. 6.6. Kod języka programowania Python, który wyświetla informację – czeka na wpis użytkownika, by pobrać dane

Wykorzystując funkcję **input()**, przesyłamy użytkownikowi komunikat, który wyświetla się w konsoli, i udostępniamy możliwość wpisania dowolnego znaku z klawiatury. Ten element języka Python jest także używany do „zatrzymania konsoli". Oznacza to, że gdy umieścimy go po ostatnim poleceniu bądź informacji dla użytkownika, konsola się nie zamknie, tylko będzie czekała na naciśnięcie klawisza [Enter].

W naszym projekcie, gdy użytkownik wpisze jakąś cyfrę, np. 5, to zostanie ona przechwycona i przypisana do zmiennej **daneInput**. Dzięki temu będziemy mogli ją wykorzystać do dalszych działań. Możesz napisać kod:

```
print(daneInput)
```

Gdy umieścisz go pod tym wskazanym na zdjęciu, wówczas wyświetli się w konsoli cyfra, którą wpisałeś. W naszym przykładzie wpisaliśmy 5 i poniżej otrzymaliśmy także piątkę.

```
Witaj serdecznie. Aby rozpocząć grę, wpi
sz liczbę z przedziału od 1 do 99, która
 Twoim zdaniem został wylosowana :
5
5
```

Rys. 6.7. Konsola Python – efekt działania funkcji **input()**

Jeżeli dodałeś jeszcze jedno polecenie **input()**, to konsola „zatrzymała się na ekranie". W innym przypadku – zaraz po wpisaniu cyfry – została zamknięta. Oczywiście możesz ten mały problem bardzo szybko naprawić, dodając na końcu wspomniany **input()**.

W następnej części podręcznika także skorzystamy z przechwyconych od użytkownika danych. Tworząc kod, oprzemy się na podstawach, które napisaliśmy w części 5.3. Zanim przejdziesz do kolejnej części, sięgnij do wspomnianego rozdziału, przypomnij sobie kody, które tam omówiliśmy.

6.4. Projekt Gra edukacyjna Zgadnij Liczbę – typy złożone i operacje na nich

W poprzednim rozdziale stworzyliśmy proste typy danych. Teraz zajmiemy się danymi złożonymi w postaci listy (pojęcie charakterystyczne np. dla języka Python, ale w innych – np. JavaScript czy PHP – są nazywane tablicami). Wykorzystamy jeszcze dwa elementy, które zostały omówione w poprzednich działach:

- funkcje,
- rekurencje.

Większą część kodu „zamkniemy wewnątrz" funkcji m.in. po to, aby móc rekurencyjnie odwoływać się do tego narzędzia programistycznego.

Nasz projekt wykonaliśmy w dwóch wersjach:

- zgadnijLiczbe.py – wersja robocza pliku,
- zgadnijLiczbeEnd.py – wersja końcowa pliku.

Na początek wprowadźmy zmianę. Spójrz na rys. 6.8. Umieść jego zawartość na górze twojego pliku. To jest część, która znajdzie się przed funkcją, a której deklarację już wykonano (na dole grafiki z kodem).

```
liczbaStartowa = 55

aLiczbaJestWieksza = "Szukana liczba jest większa,
spróbuj ponownie wpisać jakąś liczbę"
bLiczbaJestMniejsza = "Szukana liczba jest
mniejsza, spróbuj ponownie wpisać jakąś liczbę"
cZgadzaSie = "Brawo, gratuluję, zgadłeś liczbę!
To faktycznie jest liczba: "
liczbaRuchowInfo = "Liczba Twoich ruchów
potrzebnych do zgadnięcia właściwej liczby, to: "

tablicaPodawanychLiczb = []

podawaneLiczby = "Podczas zgadywania podałeś
liczby: "

daneInput = int(input("Witaj serdecznie. Aby
rozpocząć grę, wpisz liczbę z przedziału od 1 do
99, która Twoim zdaniem została wylosowana : "))

def sprawdzLiczbe(daneInput):
```

Rys. 6.8. Projekt Gra edukacyjna Zgadnij Liczbę – kod początkowy napisany w języku programowania Python

Teraz zajmijmy się funkcją **sprawdzLiczbe()**. Napisana wcześniej logika opierała się na konstrukcji warunkowej:

```
if warunek:
   wykonaj polecenie A
elif warunek:
   wykonaj polecenie B
else:
   jeżeli nieobsłużone przez if lub elif, to wykonaj polecenie C.
```

Poniżej znajdziesz grafiki kolejnych fragmentów kodu, rozpoczynając od definicji funkcji **sprawdzLiczbe()**. Wszystkie elementy kodu opisano, aby je łatwiej zrozumieć.

```
def sprawdzLiczbe(daneInput):

    # Metoda append dołącza do tablicy
    tablicaPodawanychLiczb cyfrę wpisaną przez
    użytkownika i przypisaną do zmiennej daneInput
    tablicaPodawanychLiczb.append(str(daneInput))
```

Rys. 6.9. Projekt Gra edukacyjna Zgadnij Liczbę – kod funkcji dołączający do tablicy **tablica PodawanychLiczb** liczbę wpisaną przez użytkownika i przypisaną do zmiennej **daneInput**

Kolejny fragment kodu rozpocznie instrukcję warunkową, która ustawia większość logiki tworzonego programu.

Python – dodawanie do listy

Korzystając z metody **append()**, dodajemy do listy kolejny element na jej końcu, np.:

```
listaTest = [1, 3, 6, 9]
listaTest.append(243)
```

print(len(listaTest)) – wydrukuje liczę elementów: 5

print(listaTest[4]) – wydrukuje jako ostatni element liczbę 243, czyli powstała lista:

[1, 3, 6, 9, 243].

Jeżeli do powyższej listy będziemy chcieli dodać element w jakimś konkretnym miejscu, dla przykładu na pozycji trzeciej, czyli o indeksie `2`, to wówczas trzeba użyć metody **insert()**:

```
listaTest.insert(2, "insert")
```

print(listaTest) – wydrukuje w konsoli:

[1, 3, 'insert', 6, 9, 243].

```
# To jest poczatek instrukcji
if daneInput == liczbaStartowa:
    # To jest początek tzw.  ciała instrukcji
    if
    # len() to funkcja pokazująca liczbę
    elementów umieszczonych w tablicy
    liczbaRuchow = len(tablicaPodawanychLiczb)
    # stworzenie pustego strigna (ciąg znaków)
    wpisywaneLiczby = " "
    # funkcja join(tablica) - zamienia tablicę
    w tzw. stringa
    wpisywaneLiczby = wpisywaneLiczby.join
    (tablicaPodawanychLiczb)
    # pokazywanie w konsoli założonych w
    projekcie wyników
    print(cZgadzaSie, daneInput, ", ",
    liczbaRuchowInfo, liczbaRuchow, ", ",
    podawaneLiczby,  wpisywaneLiczby)
    input("Brawo! Zadanie wykonane. Kliknij
    klawisz ENTER, aby zamknąć konsolę :).")
    # A to jest koniec pierwszego bloku (tzw.
    ciała) instrukcji if
```

Rys. 6.10. Projekt Gra edukacyjna Zgadnij Liczbę – kod funkcji (początek instrukcji warunkowej)

Podczas tworzenia projektów programistycznych warto pamiętać o zaznaczaniu istotnych elementów kodu. Dotyczy to np. jednorodnych bloków, czyli takich, które odpowiadają za określoną część funkcjonalności. Na rys. 6.10 zrealizowano to za pomocą dwóch komentarzy:

"# To jest początek instrukcji"
"# A to jest koniec pierwszego bloku (tzw. ciała) instrukcji if".

```
# Zaczynamy drugi blok
elif daneInput < liczbaStartowa:
    # To jest początek drugiego tzw. ciała
    instrukcji if, wykona się ta część kodu,
    gdy nie wykona się instrukcja po if lub jak
    wolisz, gdy wpisana liczba będzie mniejsza
    od liczby wskazanej na początku tego
    prostego algorytmu
    # pokazanie użytkownikowi w konsoli
    informacji o tym, że nie wprowadził
    właściwej liczby
    print(aLiczbaJestWieksza)
    # ponowna prośba o wprowadzenie kolejnej
    (innej) cyfry
    daneInput = int(input('Wpisz inną liczbę z
    przedziału od 1 do 99 : '))
    # uruchomienie funkcji z jej wnętrza, czyli
    tak zwana referencja
    sprawdzLiczbe(daneInput)
    # A to jest koniec drugiego bloku (tzw.
    ciała) instrukcji if
```

Rys. 6.11. Projekt Gra edukacyjna Zgadnij Liczbę – kod funkcji, instrukcja warunkowa, element **elif**

Na rys. 6.11 zwróć uwagę na podkreślenie tych elementów przez opisy:

"# Zaczynamy drugi blok"
"# A to jest koniec drugiego bloku (tzw. ciała) instrukcji if".

Na rys. 6.12 znajdziesz zaś opisy:

"# Zaczynamy trzeci blok"
"# A to jest koniec trzeciego bloku (tzw. ciała) instrukcji if".

```
# Zaczynamy trzeci blok
else:
    # To jest początek trzeciego bloku, który
    wykona się jedynie wtedy, gdy nie wykona się
    żaden z pozostałych
    # pokazanie użytkownikowi w konsoli
    informacji o tym, że nie wprowadził
    właściwej liczby
    print(bLiczbaJestMniejsza)
    # ponowna prośba o wprowadzenie kolejnej
    (innej) cyfry
    daneInput = int(input('Wpisz inną liczbę z
    przedziału od 1 do 99 : '))
    # uruchomienie funkcji z jej wnętrza, czyli
    tak zwana referencja
    sprawdzLiczbe(daneInput)
    # A to jest koniec trzeciego bloku (tzw.
    ciała) instrukcji if
```

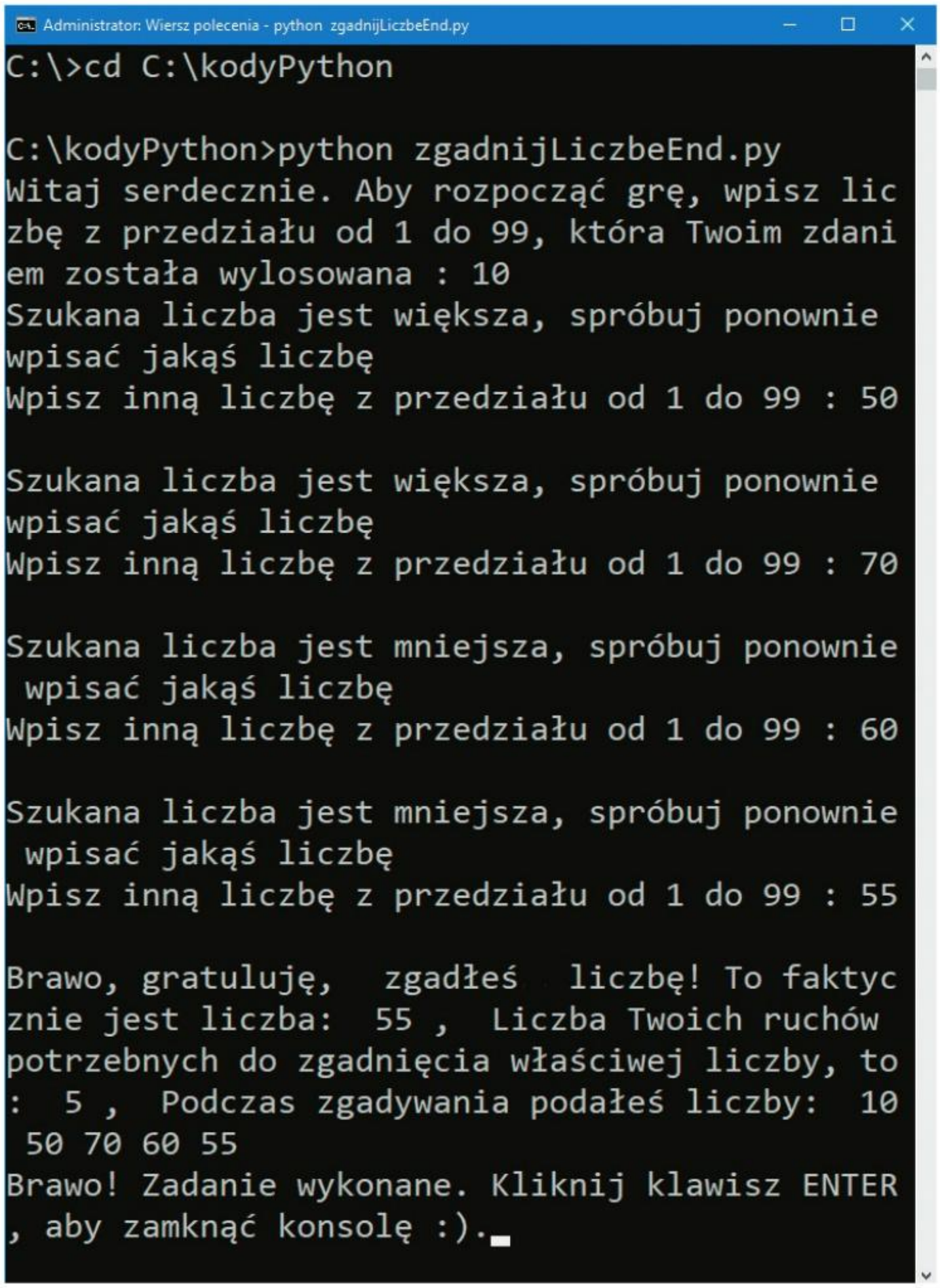
Rys. 6.12. Projekt Gra edukacyjna Zgadnij Liczbę – kod funkcji, ostatni element instrukcji warunkowej **`else`**

Na zakończenie pozostaje dodanie ostatniego elementu kodu, który uruchamia cały program – wywołanie funkcji **`sprawdzLiczbe()`**.

```
# Wywołanie funkcji z parametrem
sprawdzLiczbe(daneInput)
```

Rys. 6.13. Projekt Gra edukacyjna Zgadnij Liczbę – kod wywołania funkcji, czyli uruchomienie całego programu

W ten sposób powstał program, który funkcjonując w pętli, działa, dopóki użytkownik nie wskaże właściwej liczby (lub nie zamknie konsoli). Zobaczmy przykładowy wynik działania programu w konsoli Pythona (rys. 6.14).



Rys. 6.14. Projekt Gra edukacyjna Zgadnij Liczbę – przykładowe wykonanie napisanego kodu w konsoli Pythona

Zakończyliśmy kodowanie i testowanie pierwszej wersji programu: Gra edukacyjna Zgadnij Liczbę. Najpierw przemyśl zapisany kod. Wprowadź w nim kilka zmian. Następnie wykonaj ćwiczenia, które znajdują się na końcu tego rozdziału.

W kolejnych częściach tego podręcznika wspólnie ponownie napiszemy kod omawianej gry, wykorzystując jedną z bibliotek Pythona – tkinter. Dzięki temu otrzymamy program okienkowy możliwy do wyeksportowania, a potem także do użycia bezpośrednio w systemie operacyjnym.

6.5. Deklarowanie własnych typów danych

W niektórych językach programowania można tworzyć własne, zwykle bardziej złożone typy danych. Tylko w niektórych, gdyż w niewielu stosuje się to w praktyce. Teoretycznie, jeżeli mamy do czynienia z językiem obiektowym, to właściwie w postaci obiektu można (prawie zawsze) stworzyć nowy, własny typ danych. Tak naprawdę ma to sens głównie w językach typowanych statycznie. Typ określa:

- wielkość pamięci zaalokowanej dla zmiennej (lub wyniku wyrażenia);
- rodzaj wartości, które mogą być przechowywane w tej zmiennej;
- sposób interpretacji wspomnianych wartości (wzorce bajtów);
- operacje, jakie mogą zostać na nich wykonane.

Przykładowo: tego typu konstrukcję można zrobić w:

- C++ – wykorzystując słowo kluczowe: class, struct, enum lub enum class;
- C# – do tworzenia własnych typów niestandardowych służą konstrukcje struct, Class, Interface i enum;
- TypeScript – interfejsy i enumeratory,
- VBA – za pomocą słowa kluczowego Type.

```cpp
#include <iostream>
using namespace std;

struct MojTyp1{
    int liczba;
    string wyraz;
};

class MojTyp2{
public:
    MojTyp2(int liczba = 0, string wyraz = ""){};
    MojTyp2(const MojTyp2& mojtyp2){};
    ~MojTyp2();

    int liczba;
    string wyraz;
};

union MojTyp3{
    int liczba;
    string wyraz;
};
```

Rys. 6.15. Przykładowe typy danych tworzone przez programistę w C++

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień i krótko opisz przynajmniej trzy proste i trzy złożone typy danych.
2. Stwórz plik Pythona, a w nim zapisz w postaci zmiennych przynajmniej trzy proste i trzy złożone typy danych.
3. Wyjaśnij działanie metody **`input()`** w języku programowania Python.
4. Poszukaj w internecie dodatkowych informacji odnośnie do deklarowania własnych typów danych w C++. Przeanalizuj znalezione przykłady.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Stwórz plik Pythona. Utwórz w nim listę dziesięcioelementową, składającą się z liczb. Stwórz trzy zmienne – przypisz do nich odpowiednio ze stworzonej listy pozycje o indeksach: 0, 4, 7. Następnie dodaj wymienione zmienne i wyświetl wynik dodawania w konsoli.
2. Dodaj w kodzie Gry edukacyjnej Zgadnij Liczbę następujące funkcjonalności:
   - na początku gry:
     - prośbę do użytkownika, aby podał swoje imię,
     - przywitanie użytkownika, uwzględniające wpisane przez niego imię;
   - w logice gry zmiany takie, które w razie podania przez użytkownika znaku niebędącego liczbą całkowitą z przedziału od 1 do 99 (czyli np. ułamkiem) spowodują wyświetlenie informacji o wprowadzeniu błędnych danych.

7 Ulepszenie projektu Gra edukacyjna Zgadnij Liczbę – wyrażenia, instrukcje i biblioteki

ZAGADNIENIA

- Podsumowanie z odrobiną teorii i drobnymi ulepszeniami
- Sprawdzenie typu – czy zmienna jest typu int
- Dołączenie pierwszej biblioteki do tworzonego projektu

7.1. Podsumowanie z odrobiną teorii i drobnymi ulepszeniami

Tworząc kod projektu Gra edukacyjna Zgadnij Liczbę, wykorzystaliśmy kilka elementów ze świata programowania. Są to m.in.:

- pseudokod,
- operatory przypisania, porównania, logiczne,
- instrukcje warunkowe,
- funkcje,
- działanie w pętli z wykorzystaniem rekurencji,
- proste typy danych,
- złożone typy danych,
- operacje na typach danych.

Jest tego całkiem dużo jak na początek, jednak projekt ma pewne mankamenty, które oczywiście można ominąć, ale tylko wtedy, gdy użytkownik chce tak zrobić (musi wprowadzić do konsoli liczbę całkowitą). Oto najważniejsze problemy zbudowanej aplikacji:

A. Brak informacji o potrzebie wprowadzenia liczby całkowitej.
Sprawdź, co stanie się w konsoli, jeśli wpiszesz liczbę dziesiętną.

B. Nieatrakcyjny podział tekstu widocznego dla użytkownika.

C. Brak ograniczenia uniemożliwiającego wprowadzenia innego znaku niż liczby całkowitej z przedziału 0–99.
Sprawdź, co stanie się w konsoli, jeśli wpiszesz literę albo wyraz.

D. Ręczne wpisywanie liczby do odgadnięcia. Proces można by zautomatyzować, aby program losowo wybierał liczbę całkowitą z przedziału 0–99.

Jeżeli sprawdziłeś program tak, jak podano powyżej, to już wiesz, że wpisanie dla przykładu ciągu znaków „Witaj, graczu” spowoduje zamknięcie konsoli.

Zastanówmy się nad dokonaniem stosownych zmian w programie celem uzyskania wyższej jakości jego działania. Ulepszenie problemu z punktu A jest najprostsze, ale także bardzo ważne. Niedokładne opisy gry wykonano z premedytacją, aby teraz móc podkreślić, jak ważne jest dokładne informowanie użytkownika o zasadach działania programu. Możesz dokonać zmiany w kodzie. Zamień kod z rys. 7.1 na ten przedstawiony na rys. 7.2.

```
daneInput = int(input('Witaj serdecznie. Aby
rozpocząć grę, wpisz liczbę z przedziału od 1 do
99, która Twoim zdaniem została wylosowana : '))
```

Rys. 7.1. Projekt Gra edukacyjna Zgadnij Liczbę – kod przed poprawką

```
daneInput = int(input("Witaj serdecznie. Aby
rozpocząć grę, wpisz liczbę całkowitą z
przedziału od 1 do 99, która Twoim zdaniem
została wylosowana : "))
```

Rys. 7.2. Projekt Gra edukacyjna Zgadnij Liczbę – kod poprawiony (zmiana wskazana kolorem)

Użytkownik został poinformowany, co dokładnie powinien zrobić. Nie gwarantuje to jednak sukcesu, ponieważ jeśli zostanie wprowadzony zły znak (np. wyraz, litera, liczba dziesiętna), to program nie zadziała. Najpierw poprawmy problem wskazany w punkcie B, czyli: „nieatrakcyjny podział tekstu widocznego dla użytkownika".

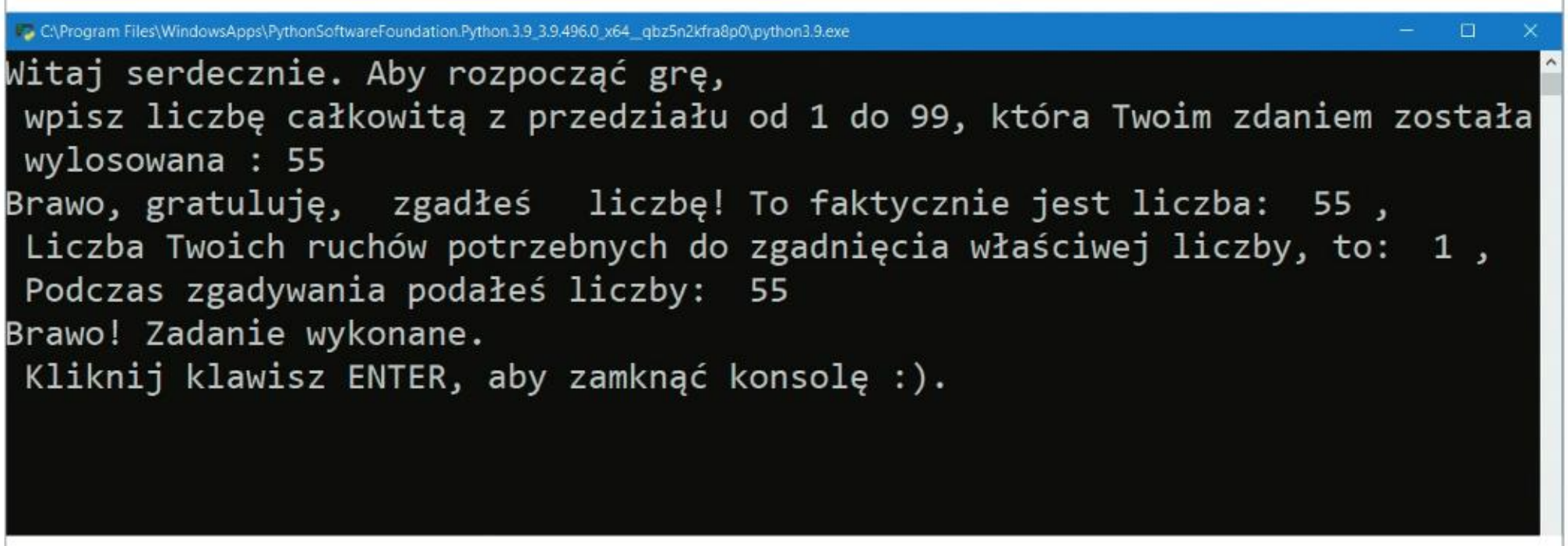
Można ładniej rozłożyć tekst, wykorzystując np. podział na nowe wiersze (w Pythonie: \n). Zmieńmy kod z rys. 7.3 na ten przedstawiony na rys. 7.4.

```
# pokazywanie w konsoli założonych w projekcie
wyników
print(cZgadzaSie, daneInput, ", ", liczbaRuchowInfo,
liczbaRuchow, ", ",podawaneLiczby,  wpisywaneLiczby)
input("Brawo! Zadanie wykonane. Kliknij klawisz
ENTER, aby zamknąć konsolę :).")
# A to jest koniec pierwszego bloku (tzw. ciała)
instrukcji if
```

Rys. 7.3. Projekt Gra edukacyjna Zgadnij Liczbę – poprawa widoku informacji kończącej grę (kod przed poprawką)

```
# pokazywanie w konsoli założonych w projekcie
wyników
print(cZgadzaSie, daneInput, ", \n",
liczbaRuchowInfo, liczbaRuchow, ", \n",
podawaneLiczby,  wpisywaneLiczby)
input("Brawo! Zadanie wykonane.\n Kliknij klawisz
ENTER, aby zamknąć konsolę :).")
# A to jest koniec pierwszego bloku (tzw. ciała)
instrukcji if
```

Rys. 7.4. Projekt Gra edukacyjna Zgadnij Liczbę – poprawa widoku informacji kończącej grę (kod po poprawce – dodane elementy zaznaczono kolorem)



Rys. 7.5. Projekt Gra edukacyjna Zgadnij Liczbę – poprawiony widok w konsoli Pythona informacji końcowej kończącej grę

7.2. Sprawdzenie typu – czy zmienna jest typu `int`

W internecie można znaleźć wiele rozwiązań dotyczących problemu: „Co zrobić, aby po wpisaniu przez użytkownika znaku innego niż liczba całkowita z przedziału od 0 do 99 pojawiła się stosowna podpowiedź zamiast natychmiastowego wyłączenia konsoli?”.

Mogłoby się wydawać, że może nam pomóc metoda **`type()`**, ale – niestety – tak nie jest. Jeżeli napiszesz poniższy kod:

```
mojaZmienna = 'jakaś zmienna'
if type(int(mojaZmienna)) == int:
    print('Zmienna jest typu int')
else:
    print('Zmienna nie jest typu int')
input(),
```

to po jego wykonaniu otrzymasz w konsoli błąd.

Z tego powodu zaproponujemy inne rozwiązanie, które opiera się na dwóch elementach:

A. Sprawdzenie, czy zmienna jest typu liczbowego
Wykorzystamy instrukcję:

`try`:

zadanie do wykonania
wynik dla wykonanego zadania

`except:`

wynik, gdy zadanie nie było możliwe do wykonania

Opisana konstrukcja programistyczna jest w języku Python wykorzystywana do obsługi błędów. Gdy po **`try`** znajdzie się coś wykonywalnego (poprawnego), to zostanie wykonane polecenie, ale jeżeli znajdzie się tam element niepoprawny (np. próba zamiany wyrazu, czyli stringa na liczbę), wówczas **`except`** obsłuży wyjątek i wykona się kod następujący po nim. Zobacz to na konkretnym przykładzie – rys. 7.6.

```
# Zmienna typu string
notNumber = "Nie jestem cyfrą :)"

# Spróbuj zamienić zmienną notNumber na liczbę całkowitą
try:
    int(notNumber)
    print("true 3")
except:
    print("false 3")
```

Rys. 7.6. Wykorzystanie instrukcji do obsługi błędów **`try - except`**

B. Sprawdzenie, czy zmienna jest typu **`int`**, czyli liczba całkowita.
Po zaprogramowaniu punktu A będziemy wiedzieli, czy wpisany przez użytkownika znak (znaki) jest typu liczbowego. Jeżeli będzie liczbowy, to w prosty sposób możemy sprawdzić, czy jest liczbą całkowitą. Spójrz na kod i jego wyjaśnienia zawarte na rys 7.7.

```
# W wyniku przyrównania liczby całkowitej z liczbą
dziesiętną  uzyskamy  fałsz, czyli w konsoli pojawi się
napis: false 1
if numberInt == numberFloat:
    print("true 1")
else:
    print("false 1")

# W wyniku przyrównania liczby całkowitej z liczbą
dziesiętną zamienioną na całkowitą  uzyskamy  prawdę, czyli
w konsoli pojawi się napis: true 2
if numberInt == int(numberFloat):
    print("true 2")
else:
    print("false 2")
```

Rys. 7.7. Wyjaśnienie sprawdzenia, czy liczba jest typu **`int`**

Spójrz jeszcze na wynik w konsoli uzyskany w rezultacie działania kodów z dwóch powyższych grafik (rys. 7.8).

```
false 3
false 1
true 2
```

Rys. 7.8. Wynik w konsoli uzyskany dzięki działaniu kodów: instrukcja do obsługi błędów **`try – except`** i sprawdzenie, czy liczba jest typu **`int`**

Tym razem napisanie kompleksowego rozwiązania będzie twoją pracą domową. Pamiętaj, że aby zostać programistą, nie można wyłącznie opierać się na cudzych zapiskach. Trzeba także na pewnym etapie nauki rozpocząć kodowanie oparte na własnych rozwiązaniach. Opierając się na podanych podpowiedziach, popraw niedoskonałości kodu programu Gra edukacyjna Zgadnij Liczbę. Możesz także poszukać w internecie alternatywnego rozwiązania i nowych funkcjonalności wykonanej gry.

7.3. Dołączenie pierwszej biblioteki do tworzonego projektu

W tym rozdziale wykonamy jeszcze jeden element programistyczny. Spowodujemy, żeby podczas każdego uruchomienia programu była generowana liczba losowa z przedziału od 0 do 99. Dotychczas przypisywaliśmy „ręcznie" cyfrę do zmiennej o nazwie liczbaStartowa. Teraz obsłużymy to prostym kodem losującym.

Do Pythona napisano tysiące różnego rodzaju bibliotek. Gdybyśmy poszukali, to co najmniej kilka z nich mogłoby dostarczyć rozwiązań zagadnienia generowania losowej liczby. W naszym projekcie wybraliśmy rozwiązanie dostarczane z głównym trzonem Pythona 9.x (The Python Standard Library).

Z dokumentacji (źródło: https://docs.python.org/3/library/random.html) możemy się dowiedzieć, że:

```
random.randint(a, b)
Return a random integer N such that a <= N <= b. Alias for
randrange(a, b+1)..
```

Zwróć uwagę na szczegóły kodu zawarte na rys. 7.9.

```
# Import biblioteki random.py
import random

# Wylosowanie liczby z przedziału 0-99
liczbaStartowa = random.randint(0,99)

# pokazanie wylosowanej liczby
print(liczbaStartowa)

# Zatrzymanie konsoli, żeby zobaczyć wynik
losowania
input()
```

Rys. 7.9. Import biblioteki random.py i kod uruchomiający losowanie liczby z przedziału od 0 do 99

Gdy wykonasz powyższy kod, zobaczysz, że w konsoli jest generowana losowa liczba. Wykorzystaj to w opracowywanym programie. Teraz już nie musisz przypisywać ręcznie cyfry do zmiennej **liczbaStartowa**.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co w programowaniu oznacza wyrażenie **\n**?
2. Wyjaśnij pojęcia: true, false.
3. Opisz, w jaki sposób można dołączać biblioteki do programów kodowanych w języku Python.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz w języku programowania Python kod sprawdzający, czy zmienna jest typu **int**.
2. Wykonaj poniższe zadania.
   A. Stwórz plik notNumber.py, a w nim zmienne: notNumber1 i notNumber2.
   B. Przypisz zmiennej notNumber1 cyfrę otoczoną cudzysłowem, np. „3”, a zmiennej notNumber2 cyfrę otoczoną cudzysłowem, np. „7”.
   C. Napisz kod, który wykona (w kolejnych linijkach) działania matematyczne na dwóch wymienionych liczbach: dodawanie, odejmowanie, mnożenie i dzielenie.
   D. Wykonaj kod zapisany w punktach A, B i C.
   E. Przedyskutuj z nauczycielem i innymi uczniami otrzymane wyniki.
3. Popraw niedoskonałości kodu programu Gra edukacyjna Zgadnij Liczbę, uwzględniając opisane w tym rozdziale podpowiedzi:
   - instrukcja do obsługi błędów **try - except**,
   - sprawdzenie, czy liczba jest typu **int**.

   Warto też poszukać rozwiązania alternatywnego do uatrakcyjnienia (utrudnienia dla gracza) stworzonej aplikacji.
4. Zaimplementuj generowanie liczby losowej do opracowywanego kodu w taki sposób, aby nie trzeba było przypisywać ręcznie cyfry do zmiennej **liczbaStartowa**, ale by automatycznie wykonywał to program podczas startu.

8 Zasady programowania obiektowego

ZAGADNIENIA

- Świat obiektów
- Podstawowe pojęcia z zakresu programowania obiektowego
- Hermetyzacja, dziedziczenie i polimorfizm
- Tworzenie klas i obiektów

8.1. Świat obiektów

Programowanie obiektowe jest prawdopodobnie najbardziej popularnym sposobem tworzenia kodów. Dzieje się tak zapewne dlatego, że otaczający nas świat składa się z obiektów, np.:

- monitor,
- laptop,
- klawiatura,
- biurko,
- myszka,
- fotel.

Można tak po kolei wymieniać zasoby pokoju, domu, otaczającego nas świata, a nawet pójść jeszcze dalej. Efektem obiektowej budowy naszego środowiska życia jest postrzeganie przez ludzi wszystkiego jako obiektów. Ty także nie jesteś zapewne wyjątkiem, nawet jeżeli jeszcze nie zdajesz sobie z tego sprawy. Ponieważ nasze mózgi na co dzień widzą obiekty, więc obiektowość jest tym, co nam jest stosunkowo łatwo zrozumieć i tego używać. Zapewne właśnie dlatego omawiany sposób programowania to podstawa w pracy dużej części programistów.

Wyjaśnijmy, na czym polega obiektowość. Wyobraź sobie zwierzę, np. rysia. Spójrz na obrazek.

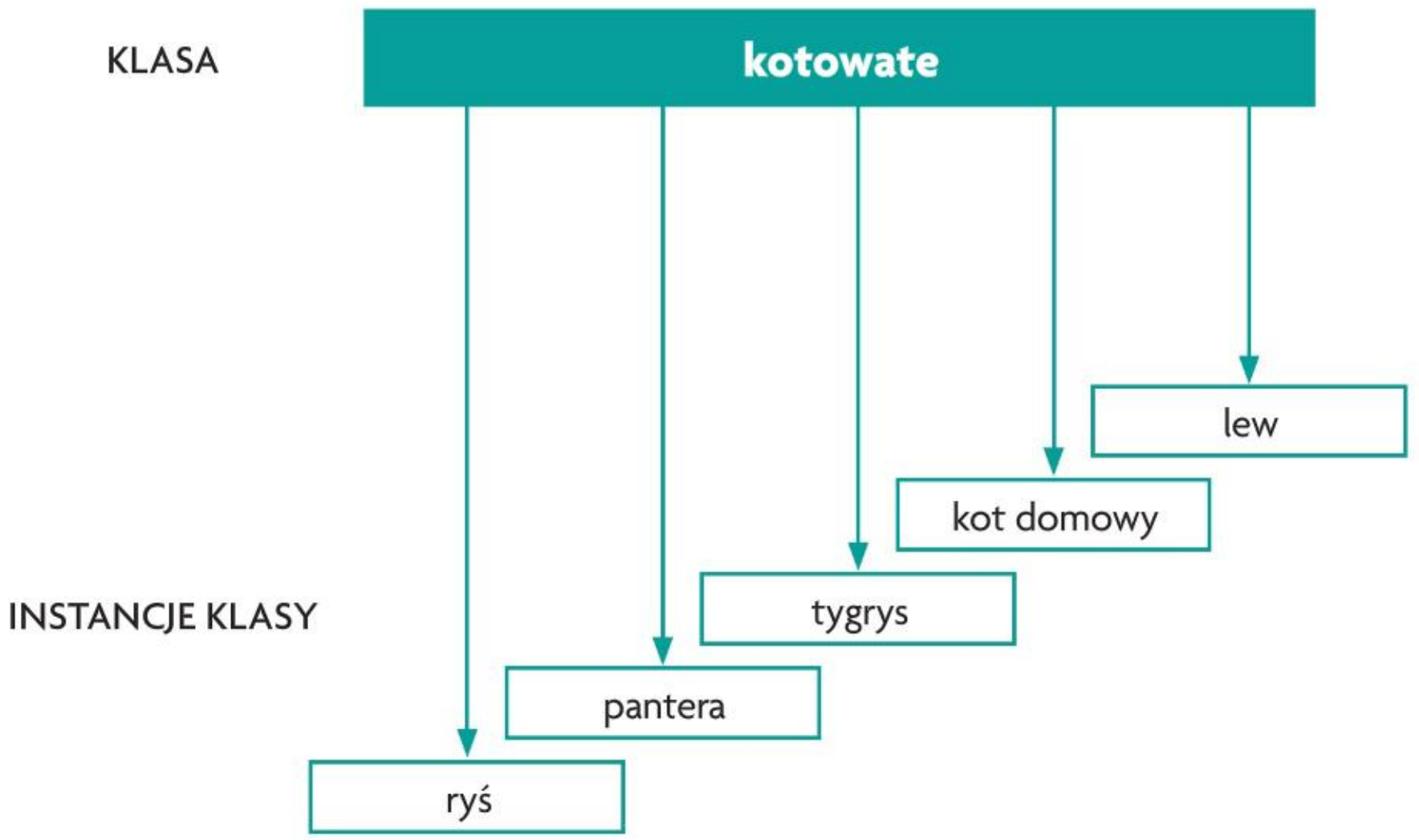
Na grafice ryś-obiekt ma:

- **właściwości** (cechy) – czyli coś statycznego, typu: oczy, nos, wąsy, sierść, ogon, określony kolor itp.
- **metody** (funkcje) – działanie, czyli elementy dynamiczne, ryś zapewne potrafi: chodzić, biegać, jeść, spać, rozszarpywać pazurami itd.



Rys. 8.1. Przykładowy obiekt

Wskazane **metody** i **właściwości** to elementy nie tylko rysia, ale także programowania obiektowego. Wydaje się to łatwe, choć problem jest bardziej skomplikowany. Przyjęliśmy za obiekt coś, co jest instancją klasy, a mamy jeszcze w programowaniu element nadrzędny, czyli klasę: **KLASA JEST SZABLONEM OBIEKTÓW, A OBIEKT JEST INSTANCJĄ KLASY.**



Rys. 8.2. Klasa jest szablonem obiektów, a obiekt jest instancją klasy

Idąc jeszcze dalej, należałoby stwierdzić, że to, co powiedzieliśmy o rysiu, jest nie do końca dobrym przykładem. Gdyby się trzymać zasady, że **klasa powinna obejmować jak najmniejszą i w miarę możliwości jednorodną grupę obiektów**, to jako:

- **klasę** – powinniśmy przyjąć **rysia jako gatunek**;
- **obiekt** – to byłby **pojedynczy egzemplarz z gatunku ryś**, czyli np. **Rysiek Kazik (RysKazik)**.

Wydaje się, że lepiej do wyobraźni przemawia jednak ryś jako kotowaty. Pomimo pewnych niedociągnięć możemy trzymać się także tej wersji. Zatem dla naszego rysia elementem nadrzędnym, czyli klasą, mogłaby być klasa zwierzęta albo kotowate. Z kolei ryś jest egzemplarzem wykonanym na bazie klasy. Zauważ, że pewne informacje o rysiu są charakterystyczne dla wszystkich zwierząt, czyli klasa kotowate miałaby:

- **właściwości** (cechy) – coś statycznego, typu: oczy, nos, wąsy, skóra, określony kolor itp.;
- **metody** (funkcje) – działanie, czyli elementy dynamiczne – zwierzę potrafi: chodzić, biegać, jeść, spać itd.

8.2. Podstawowe pojęcia z zakresu programowania obiektowego

Na początek przypomnijmy sobie:

- czym jest programowanie obiektowe;
- jakie są główne cechy tego stylu programowania.

Jeżeli dopiero poznajesz żargon programowania obiektowego, to koniecznie musisz poznać kilka pojęć podstawowych dla tego obszaru:

- obiekt (instancja klasy) – metoda, pole (właściwość),
- klasa,
- hermetyzacja,
- agregacja,
- dziedziczenie – ponowne wykorzystanie,
- polimorfizm.

Obiekt – metoda, pole (właściwość)

Obiekt w języku programowania obiektowego reprezentuje określony byt (organizm żywy lub rzecz):

- element świata fizycznego lub
- twór abstrakcyjny.

Obiekt:

- ma charakterystyczne cechy – nazywane w programowaniu właściwościami;
- może wykonać określone czynności – nazywane w programowaniu metodami.

Posługując się terminologią programowania obiektowego, możemy dla lepszego zrozumienia wytłumaczyć pewną analogią do języka mówionego:

- do nazwania obiektu wykorzystuje się zwykle rzeczowniki (ryś, pies, telewizor, wojownik, czarownik, wiedźma, nauczyciel, uczeń itp.);

- dla metod charakterystyczne są czasowniki (np. walczy, prezentuje, czaruje, chodzi, ryczy, naucza, uczy się);
- właściwości (a dokładniej ich wartości) to przymiotniki.

```
// Obiekt JavaScript: blok DIV, zawiera
jedynie właściwości
var divTestObj = {
  tekstTest: "Technologie internetowe -
  usługi IT",
  tekstPTest: "Witaj na stronie naszej firmy!
  ",
  linkComp: "https://fajnaFirma.bok/",
  obrazekTest: "images/logo_fajne.png"
}
```

Rys. 8.3. Przykładowy obiekt zawierający właściwości (cechy) – język programowania JavaScript

Klasa

Grupowanie to czynność świata rzeczywistego, która pozwala utrzymać określony porządek. Wykonuje się ją według pewnych kryteriów. To zagadnienie bywa bardzo różnie rozumiane. Wróćmy do obiektu ryś. Jeden z podziałów czworonogów wskazuje na (co najmniej) dwie grupy:

- psowate – pies domowy, wilk, lis, likaon, kojot itp.,
- kotowate – ryś, kot domowy, pantera, lew, tygrys.

Psowate i kotowate mogą jednak należeć do jednej grupy o nazwie ssaki. W powyższych opisach nazwa nadrzędna wynikająca z jakiegoś grupowania, czyli ssak, psowate, kotowate, może być klasą. Przy czym :

- w życiu codziennym najpierw ustalamy pewne cechy i działanie (funkcje) i na tej podstawie dokonujemy grupowania na zasadzie podobieństwa;
- w programowaniu najpierw tworzymy klasę, a dopiero potem jej egzemplarze, czyli obiekty.

```
var Person = {
    firstName: 'Jan',
    lastName: 'Kowalski',
    birthDate: new Date('1971-10-20'),
    gender: 'mężczyzna',
    getAge: function() {
        var today = new Date();
        var diff = today.getTime() - this.
        birthDate.getTime();
        var year = 1000 * 60 * 60 * 24 * 365.25;
        return Math.floor(diff / year);
    }
}
```

Rys. 8.4. Przykładowy obiekt zawierający metodę getAge i kilka właściwości (cechy) – język programowania JavaScript

8.3. Hermetyzacja, dziedziczenie i polimorfizm

8.3.1. Hermetyzacja

Wspomniano już o obiekcie telewizor. Przypomnijmy, że ma on:

- właściwości – kolor, rozmiar itp.,
- metody – włącz, wyłącz itp.

Większość informacji o nim, np.:

- zasady działania – skąd biorą się ruchome obrazki na ekranie (a nawet co to jest ekran),
- skład – zawartość bliżej nam nieznanych (i w zdecydowanej większości dla nas niezrozumiałych) elementów w czarnej puszce z jakiegoś tworzywa,

jest ukryta. I to jest właśnie **hermetyzacja**. Nie musimy wiedzieć o tych elementach prawie nic. Jedyne, co potrzebujemy, to umieć używać:

- przycisków, które są na nim (telewizorze),
- pilota.

```php
class TaskController extends Controller
{
    public function index()
    {
        $tasks = Task::where("iscompleted",
        false)->orderBy("id", "DEC")->get();

        $c_tasks = Task::where("iscompleted",
        true)->get();
        return response()->json(['tasks' =>
        $tasks, 'c_tasks' => $c_tasks]);
    }
}
```

Rys. 8.5. Przykładowa klasa zawierająca publiczną metodę index – język programowania PHP

Tak samo jest w programowaniu obiektowym. Otrzymujemy dostęp do publicznych metod i korzystamy z dobrodziejstwa, np. obiektów zawartych w bibliotekach programistycznych. Nie wiemy, jak są zbudowane, bo tak naprawdę nie jest nam ta wiedza potrzebna. Mamy tylko instrukcję, co robią i jak ich używać. Korzystamy z nich w podobny sposób jak z pilota telewizora. Choć warto od razu wspomnieć, że w niektórych językach programowania metody i właściwości mogą mieć dodatkowe cechy powodujące, że są one:

- publiczne (ang. *public*),
- prywatne (ang. *private*),
- chronione (ang. *protected*).

Dodatkową kategoryzację wprowadza się ze względu na różny poziom dostępu użytkownika do poszczególnych części obiektu. Przykładowo: metody prywatne mogą być dostępne tylko dla wewnętrznej implementacji obiektu, a publiczne – dla każdego. Więcej na ten temat znajdziesz w kolejnych rozdziałach podręcznika.

Bardzo często na hermetyzację programiści potocznie mówią enkapsulacja – od angielskiego słowa *encapsulation*.

8.3.2. Dziedziczenie (ponowne wykorzystanie)

Dziedziczenie w programowaniu działa podobnie jak w codziennym życiu. Dzieci dziedziczą po rodzicach majątek, ale także pewne określone cechy (kolor oczu, włosów, rysy

twarzy) i działania (szybki sposób przemieszczania się, określony sposób wypowiadania myśli). Z kolei nasz RysKazik po rysiach dziedziczy m.in. wąsy, sierść, rozszarpywanie pazurami. Obiekt ryś bierze zaimplementowane w rysiach określone funkcjonalności i właściwości.

Rys. 8.6. Dziedziczenie w języku CSS – widok konsoli dewelopera przeglądarki FireFox

8.3.3. Polimorfizm

Wyobraź sobie, że tworzysz kod, w którym dla dwóch kotowatych – np. kota domowego i tygrysa – wykorzystujesz metodę rozszarpywanie pazurami. Zapewne każdy z tych obiektów wykona tę metodę nieco inaczej, choćby ze względu na swoje gabaryty. Metoda jest jednak ta sama i to właśnie jest **polimorfizm**. Podsumujmy – mamy z nim do czynienia, gdy **na różnych obiektach możemy wywołać tę samą metodę**. Aby to jeszcze lepiej zobrazować, pomyśl o metodzie jedź. Gdy otrzyma ją obiekt samochód, to uruchomi silnik i się przemieści. Tak samo postąpi obiekt rower – zmieni miejsce swojego położenia, tylko że w nieco inny sposób niż samochód.

8.3.4. Agregacja

W programowaniu obiektowym bardzo często rozwiązanie zadania składa się z połączenia kilku obiektów. Przeprowadza się agregację, czasem nazywaną kompozycją. W programowaniu istnieje przekonanie, że jedną z najlepszych metod rozwiązywania skomplikowanych problemów jest postępowanie według zasady „dziel i zwyciężaj".

Z tego względu zagadnienia są dzielone na małe kawałki, które po rozwiązaniu łączy się w całość. W programowaniu wiele języków stosuje modułowość, np. JavaScript (szczególnie najpotężniejsze frameworki typu React, nodejs, Angular) i Python. Rozwiązując małe problemy, łączymy wszystko w jednym pliku.

```
import React, { Component } from "react";
import MenuButton1 from "./MenuButton1";
import MenuButton2 from "./MenuButton2";
import Menu1 from "./Menu1";
import Menu2 from "./Menu2";
```

Rys. 8.7. Agregacja – przykład importu modułów (plików JavaScript), aplikacja oparta na React i ES6

8.4. Tworzenie klas i obiektów

Jako podsumowanie tej części programowania obiektowego stworzymy w języku Python klasę rysie.

```
class Rysie:
    # Proste cechy (właściwości, pola)
    cecha1 = "kotowaty"
    cecha2 = "nos"
```

```
    # Prosta metoda
    def opis(self):
    print("Ja jestem", self.cecha1)
print("Ja mam", self.cecha2)
```

Następnie doprowadzimy do powstania instancji obiektu:

```
RysKazik = Rysie()
```

W ostatnim kroku uzyskamy dostęp do metody i właściwości nowo stworzonej instancji klasy rysie o nazwie RysKazik:

```
print(RysKazik.cecha1)
RysKazik.opis()
```

Całość kodów znajdziesz na rys. 8.8.

```
# Tworzenie instancji w języku Python

class Kotowate:
    # Proste cechy (właściwości, pola)
    cecha1 = "kotowaty"
    cecha2 = "nos"

    # Prosta metoda
    def opis(self):
        print("Ja jestem", self.cecha1)
        print("Ja mam", self.cecha2)

# Tworzenie instancji obiektu
Rys = Kotowate()

# Dostęp do cecha i metod klasy poprzez
obiekty
print(Rys.cecha1)
Rys.opis()
```

Rys. 8.8. Kod języka programowania Python – klasa kotowate i jej egzemplarz ryś

Po uruchomieniu powyższego kodu w konsoli Pythona ukażą się napisy takie jak na rys. 8.9.



Rys. 8.9. Wynik w konsoli zastosowania kodu języka programowania Python – klasa rysie i jej egzemplarz ryś

SPRAWDŹ SWOJĄ WIEDZĘ

1. Opisz własnymi słowami wybrany obiekt świata rzeczywistego. Wskaż jego właściwości i metody.
2. Wyjaśnij pojęcia:
   - klasa,
   - obiekt,
   - metoda,
   - pole,
   - dziedziczenie,
   - hermetyzacja,
   - agregacja,
   - polimorfizm,
   - instancja klasy.
3. Czym różni się klasa od obiektu?
4. Każdy język programowania ma specjalne słowo dotyczące klasy, które czasem programiści określają jako „magiczne”. W wielu językach programowania jest to „this”, lecz nie w Pythonie. W tym języku programowania mamy do czynienia ze słowem „self”. Znajdź w internecie informacje na temat tego, co można uzyskać w programowaniu obiektowym za pomocą słowa „self”. Opisz tę kwestię w kontekście:
   - wnętrza klasy,
   - obiektu, który jest tworzony z wykorzystaniem tego słowa.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Stwórz plik pierwszaKlasa.py i napisz w nim w języku programowania Python kod, który stworzy dowolną klasę zawierającą:
   a) co najmniej jedną właściwość,
   b) co najmniej jedną metodę.

   Na podstawie stworzonej klasy:
   a) utwórz jej instancję, czyli obiekt,
   b) na stworzonym obiekcie wywołaj zakodowane w klasie metodę i właściwość.

   Wynik pracy wyświetl w konsoli za pomocą polecenia **`python pierwszaKlasa.py`**.

9 Klasy i ich wykorzystanie w programowaniu

ZAGADNIENIA

- Zakres widoczności metod, pól klasy – kwalifikatory dostępu
- Deklaracja klasy, konstruktory i destruktory
- Tworzenie obiektów na bazie klasy
- Klasy dziedziczone i zaprzyjaźnione
- Składnik statyczny klasy i metody ich obsługi

9.1. Zakres widoczności metod, pól klasy – kwalifikatory dostępu

W nowoczesnym programowaniu istnieje kilka podstawowych metod tworzenia kodu. Jednym z najbardziej rozpowszechnionych i popularnych jest programowanie obiektowe.

Wewnątrz elementu określanego jako klasa programiści zamykają określone:

- właściwości (pola),
- metody (funkcjonalność).

Użytkownikom, którzy korzystają z danej klasy, zwykle udostępnia się jedynie jej określone elementy. Jest to związane z co najmniej dwoma kwestiami:

- nie ma potrzeby, aby każdy (nawet inny programista), który będzie używał danej klasy, rozumiał wszystko to, co jest zaprogramowane w klasie, w związku z czym wprowadzanie zmian w klasie nie zawsze musiałoby się zakończyć pozytywnym efektem;
- użytkownik klasy korzysta z jej dobrodziejstw, np. w postaci określonego interfejsu (użyteczności, funkcjonalności), a nie z klasy jako całości.

Aby sprostać powyższym założeniom, w językach programowania stworzono różnorodne mechanizmy. Jednymi z najbardziej skutecznych i najczęściej stosowanych w programowaniu aplikacji są specyfikatory.

Specyfikator lub inaczej **modyfikator** (czasem pojawia się także operator widoczności) dostępu określa poziom dostępu do poszczególnych składowych klasy. W wielu językach programowania (np. C++, PHP, C#, Java) występują trzy rodzaje tego typu elementów:

- public,
- private,
- protected.

Public

Słowo kluczowe **public** zezwala na dostęp do stałych, zmiennych, metod itp. z poziomu klasy i będąc poza nią.

```cpp
#include <iostream>

// Stworzenie klasy nazwaPoziomu
class nazwaPoziomu
{
// ustawienie poziomu dostępu na public
public:
    int liczbaWojownikow;
    nazwaPoziomu()
        : liczbaWojownikow( 0 )
    { }
    void dodajWojownika( const char * nowy )
    {
        liczbaWojownikow++;
    }
};

// Wywołanie głównej funkcji main()
int main()
{
    // ustawienie "polskich ogonków"
    setlocale(LC_ALL, "");
    // Stworzenie instancji klasy
    nazwaPoziomu posiadaniWojownicy;
    posiadaniWojownicy.dodajWojownika( "Kotowaty"
    );
    posiadaniWojownicy.dodajWojownika( "Psowaty" )
    ;

    // pokazanie w konsoli liczby stworzonych
    obiektów
    std::cout << "Może używać: " <<
    posiadaniWojownicy.liczbaWojownikow << "
    wojowników";
    return 0;
}
```

Rys. 9.1. Kwalifikatory dostępu public kod Cpp.png

Private

Słowo kluczowe **private** zezwala na dostęp do stałych, zmiennych, metod itp. jedynie z poziomu danej klasy. Podczas próby uzyskania dostępu do prywatnej metody czy zmiennej, będąc poza klasą, kompilator poinformuje o błędzie. Dostęp do metod i pól prywatnych poza klasą możemy uzyskać tylko w klasach i funkcjach zaprzyjaźnionych.

```
class nazwaPoziomu
{
private:
    int liczbaWojownikow;
public:
    nazwaPoziomu()
        : liczbaWojownikow( 0 )
    { }
    void dodajWojownika( const char * nowy )
    {
        liczbaWojownikow++;
    }

    int liczba_wojownikow() const
    {
        return liczbaWojownikow;
    }
};
```

Rys. 9.2. Kwalifikatory dostępu private – kod w języku C++

Protected

Słowo kluczowe **protected**, podobnie jak private, nie zezwala na dostęp do stałych, zmiennych, metod, będąc poza klasą. Składowe klasy oznaczone jako protected:

- są widoczne w obrębie klasy pochodnej;
- mają dostęp chroniony bądź prywatny w klasie pochodnej w zależności od użytego prawa dostępu podczas dziedziczenia;
- są widoczne w klasie / funkcji zaprzyjaźnionej.

```
class nazwaPoziomu {
protected:
    int liczbaWojownikow;
    double cena;
public:
    nazwaPoziomu();
};

// Wyzerowanie zmiennych
nazwaPoziomu::nazwaPoziomu()
    : liczbaWojownikow( 0 )
    , cena( 0 )
{
};
```

Rys. 9.3. Kwalifikatory dostępu protected – kod w języku C++

Brak kwalifikatora

W przypadku braku operatora w C++ wszystkie pola i metody w klasie są widoczne jako prywatne. Dostęp do nich z zewnątrz mają wyłącznie klasy i funkcje zaprzyjaźnione.

Zakresy widoczności podczas dziedziczenia

Dziedziczenie pól i metod po rodzicu może zmienić ich zakres. Mimo że dziedziczenie dokładniej omówimy w następnym rozdziale, warto już teraz zaznajomić się z zasadami działania systemu zakresów dziedziczenia w C++.

Dziedziczenie przy public oznacza, że dziedziczone elementy (np. zmienne lub funkcje) mają taką widoczność jak w klasie bazowej, czyli:

- gdy było public, to pozostaje public;
- gdy było protected, to pozostaje protected;
- gdy było private, to brak dostępu w klasie pochodnej.

Dziedziczenie przy private:

- gdy było public, to zmienia się na protected;
- gdy było protected, to pozostaje protected;
- gdy było private, to brak dostępu w klasie pochodnej.

Dziedziczenie przy protected:

- gdy było public, to zmienia się na private;
- gdy było protected, to pozostaje private;
- gdy było private, to brak dostępu w klasie pochodnej.

9.2. Deklaracja klasy, konstruktory i destruktory

W rozdziale 8.3 utworzyliśmy klasę w języku Python, a w niej umieściliśmy dwa pola i metodę. Wykonaliśmy to w najprostszy sposób.

```
class Kotowate:
    # Proste cechy (właściwości, pola)
    cecha1 = "kotowaty"
    cecha2 = "nos"

    # Prosta metoda
    def opis(self):
        print("Ja jestem", self.cecha1)
        print("Ja mam", self.cecha2)
```

Rys. 9.4. Kod języka programowania Python – stworzenie klasy kotowate

Dużo częściej w świecie programowania wykorzystuje się **konstruktor**. Służy on do konstruowania, a skoro coś stworzymy, to można zapewne dokonać także zniszczenia, czyli musi być dostępny również **destruktor**. W praktyce, tworząc klasę, bardzo często chcielibyśmy, aby zostały wykonane niezbędne instrukcje na początku lub na końcu istnienia obiektu. Do tego służą konstruktory i destruktory.

W różnych językach programowania konstruktory i destruktory są nieco odmiennie tworzone i rozumiane.

Konstruktor to funkcja w klasie, która jest wywoływana podczas tworzenia każdego obiektu danej klasy. Kluczowe jest tutaj słowo „każdego", czyli zawsze zostanie ona wywołana podczas powstawania obiektu. W klasie może być kilka konstruktorów.

Destruktor jest funkcją, która jest wykonywana w celu zwolnienia pamięci przydzielonej określonym zasobom (obiektom lub innym).

Przyjrzyjmy się dwóm przykładowym językom programowania: C++ i Python.

C++ – konstruktor i destruktor

Funkcja konstruktor w C++ musi spełnić warunki:

- ma identyczną nazwę jak nazwa klasy, w której jest zapisana;
- nie zwraca żadnej wartości (nawet void).

```
class Doba
{
    public:
        int liczbaGodzin;
        int liczbaMinut;
        // Konstruktor w C++
        Doba(int miesiac, int rok);
        // Destruktor w C++
        ~Doba();
};
```

Rys. 9.5. Kod języka programowania C++ – proste konstruktor i destruktor

Z kolei destruktor jest wywoływany automatycznie podczas niszczenia obiektów. Nie przyjmuje żadnych argumentów. O ile konstruktorów w danej klasie może być dowolnie dużo, to destruktor może być tylko jeden.

Jednym z ciekawszych rozwiązań C++ jest **konstruktor kopiujący**, którego głównym zadaniem jest utworzenie obiektu na bazie innego obiektu (skopiowanie jego pola). Jeżeli w kodzie nie zaimplementujemy konstruktora kopiującego, to kompilator i tak go stworzy – zrobi to automatycznie. Konstruktor kopiujący będzie po prostu tworzył drugą instancję wszystkich pól obiektu.

Python – konstruktor i destruktor

W języku programowania Python klasy nie mają wyraźnie sprecyzowanych konstruktorów i destruktorów. Dysponują za to konstrukcją, która przypomina i działa jak wyżej wymienione elementy w C++. Dotyczy to metod:

- **`__init__`** – można ją uznać za konstruktora;
- **`__del__`** – wykonuje pracę podobną dla typowego programistycznego destruktora.

W następnych rozdziałach będziemy opierali budowę aplikacji mobilnych i desktopowych na powyższych metodach.

```
class Samochod:
    def __init__(self):
        print('Samochód został stworzony.')

    def __del__(self):
        print('Destruktor: Samochód został
        zniszczony.')

# Stworzenie obiektu z wykorzystaniem
konstruktora
car = Samochod()
# Zniszczenie obiektu z wykorzystaniem
destruktora
del car

input()
```

Rys. 9.6. Kod języka programowania Python – proste konstruktor i destruktor

Zniszczenie obiektu w Pythonie jest opcjonalne. Można tworzyć obiekty i nigdy ich nie usuwać. Zostaną one skasowane automatycznie tylko po zamknięciu programu. Ze względów wydajnościowych, szczególnie w dużych programach, nieusuwanie obiektów może być problemem. W skrajnym przypadku „pamięciożerne" obiekty mogą uniemożliwić funkcjonowanie całej aplikacji. W Pythonie można zaimplementować destruktor metodą **`__del__`**.

9.3. Tworzenie obiektów na bazie klasy

Znamy już metodę **`init()`** i podstawy tworzenia konstruktorów. Pora wykorzystać nabytą wiedzę w praktyce. Wrócimy do kodu stworzonego w rozdziale 8.3. Przerobimy kod, używając konstruktora i destruktora. Na rys. 9.7 znajdziesz zapisy w pliku napisanym za pomocą języka programowania Python, uwzględniające wiedzę nabytą przed chwilą.

```
# Wykorzystanie konstruktorów i destruktorów
w języku Python

class Kotowate:
    # Proste cechy (właściwości, pola)
    cecha1 = "kotowaty"
    cecha2 = "nos"

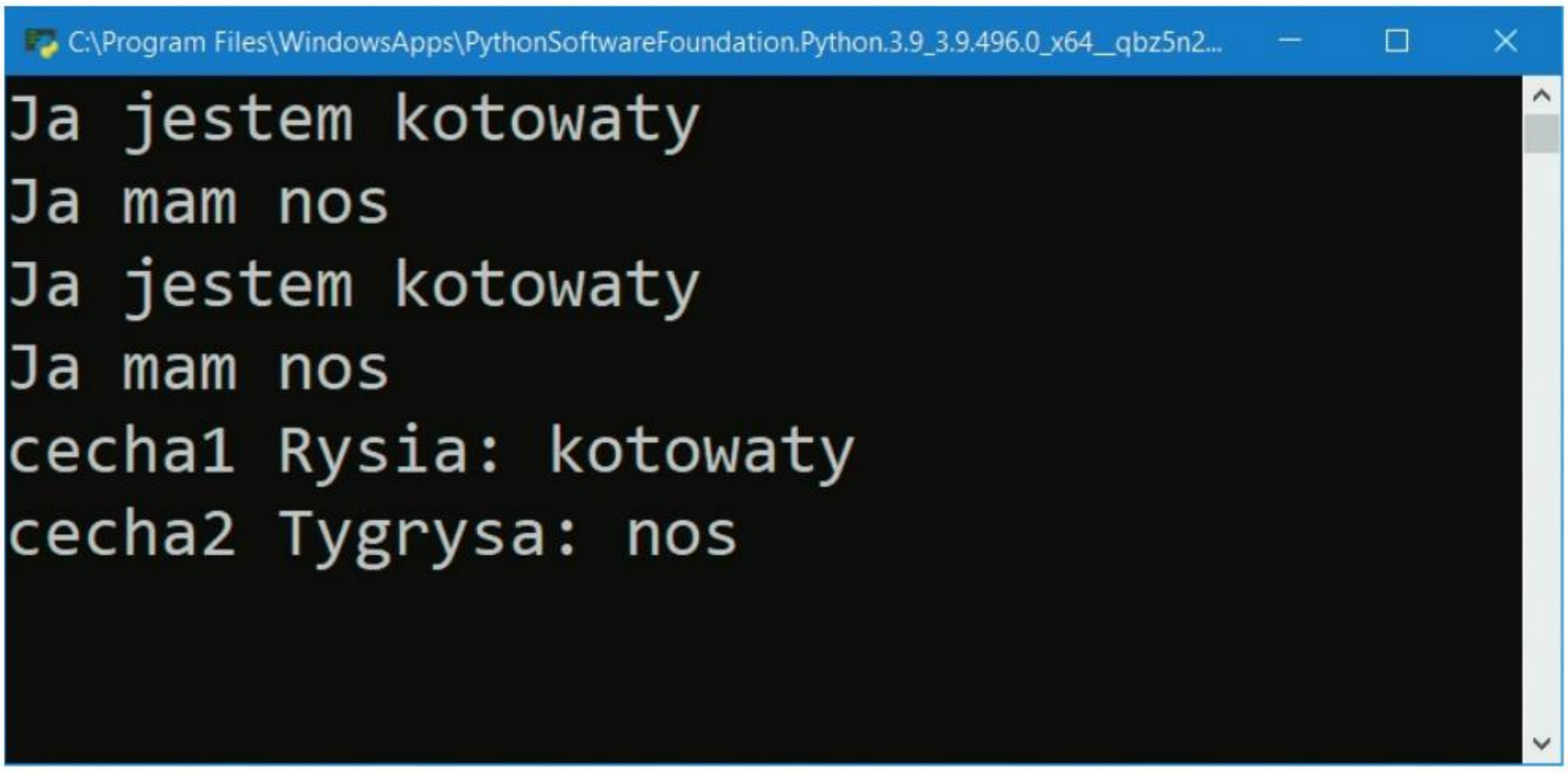
    # Konstruktor - prosta metoda
    def __init__(self):
        print("Ja jestem", self.cecha1)
        print("Ja mam", self.cecha2)

    # Destruktor - wywołanie
    def __del__(self):
        print("Wywołany destruktor")

# Tworzenie instancji obiektu
# Konstruktor powoduje automatyczne
'odpalenie' funkcji init()
Rys = Kotowate()
Tygrys = Kotowate()
# Dostęp do cechy poprzez obiekty
print("cecha1 Rysia: " + Rys.cecha1)
print("cecha2 Tygrysa: " + Tygrys.cecha2)
```

Rys. 9.7. Kod języka programowania Python – konstruktor i destruktor

Pokazany kod działa nieco inaczej niż ten poprzedni, który nie zawierał konstruktora i destruktora. Wynikiem utworzenia nowej instancji klasy kotowate w postaci obiektów ryś i tygrys jest od razu uruchomienie zawartości funkcji inicjalizującej **`__init__(self)`**. Sposób dostępu do pól klasy pozostał taki sam, czyli **`Rys.cecha1`**, a dla drugiego obiektu – **`Tygrys.cecha2`**. Wynik kodu uzyskany w konsoli znajdziesz na rys. 9.8.



Rys. 9.8. Kod języka programowania Python – konstruktor i destruktor

9.4. Klasy dziedziczone i zaprzyjaźnione

Na ostatniej lekcji pokazaliśmy, jak utworzyć własny typ danych – klasę. Aby obiekt kot miał inne możliwości niż obiekt tygrys, można napisać dwie klasy. Nie jest to jednak dobre rozwiązanie. Przecież wiemy, że tygrys i kot mają wspólne atrybuty, więc nie warto pisać część kodu dwukrotnie. Dużo lepszym rozwiązaniem będzie odziedziczenie części możliwości od klasy kotowate, a następnie sprecyzowanie ich atrybutów w klasach dziedziczących.

Aby dziedziczyć w C++, używamy następującej struktury w nagłówku klasy:

```
class Nazwa_klasy_nowej : [operator_zakresu_dziedziczenia]
Nazwa_klasy_bazowej
```

Dla rysia, tygrysa i kotowate będzie to wyglądać tak, jak przedstawiono na rys. 9.9 i 9.10.

Przypatrz się uważnie poniższym kodom. Możesz zauważyć, że klasa „Rys" jako dziecko wywołuje w swoim konstruktorze konstruktor rodzica – z użyciem wywołania w postaci **"Rys(): Kotowate(){…}"**.

W konstruktorze kopiującym konstruktor kopiujący rodzica wywołuje w taki sam sposób jak domyślny konstruktor (tylko z odpowiednim argumentem w nawiasie, który jest stałą referencją typu „Rys"). Wywoływanie odpowiadającego konstruktora rodzica przez konstruktor dziecka jest zawsze wymagane, aby w bardziej złożonych projektach nie stracić pamięci do elementów zawartych w pamięci klasy rodzica.

Destruktor klasy potomnej nigdy nie wywołuje destruktora rodzica – w naszym przypadku kompilator C++ robi to za nas. Destruktora w języku C++ nie powinniśmy nigdy sami wywoływać – debugger nie zwróci nam błędu, ale kompilacja programu może zakończyć się błędami pamięci.

```cpp
#include <iostream>
using namespace std;

class Kotowate {
public:
    //Konstruktor bezparametrowy
    Kotowate() { cout << "Tworze Kotowaty" << endl; };
    //Konstruktor kopiujący
    Kotowate(const Kotowate& kotowate) {};
    //Destruktor
    ~Kotowate() { cout << "Usuwam Kotowaty" << endl; };
};

class Rys : public Kotowate {
public:
    Rys() :Kotowate() { cout << "Tworze Rysia" << endl; };
    Rys(const Rys& rys) :Kotowate(rys) {};
    ~Rys() { cout << "Usuwam Rysia" << endl; };
    void dajGlos() {
        cout << "Miauuu" << endl;
    }

};
```

Rys. 9.9. Kod języka programowania C++ – proste dziedziczenie (część 1)

Na przedstawionych przykładach klasa „Tygrys" wykorzystuje klasę rodzica „Kotowate" w ten sam sposób, to znaczy wywołuje w odpowiednich miejscach konstruktor i konstruktor kopiujący rodzica, a destruktor dziecka nie wywołuje destruktora swojego rodzica.

Po uruchomieniu całego kodu powinieneś zauważyć, tak jak pokazano na rys. 9.11, że swoje działanie rozpoczynają najpierw konstruktory rodzica, a dopiero później odpowiednie konstruktory dzieci. Co ważne, destruktory najpierw startują od dzieci, a dopiero na końcu destruktory rodzica. Zrozumienie kolejności wywołań odpowiednich konstruktorów i destruktorów jest bardzo przydatne w programowaniu obiektowym i ułatwia programowanie oraz debugowanie kodu.

```
class Tygrys : public Kotowate {
public:
    Tygrys() :Kotowate() { cout << "Tworze Tygrysa" << endl; };
    Tygrys(const Tygrys& tygrys) :Kotowate(tygrys) {};
    ~Tygrys() { cout << "Usuwam Tygrysa" << endl; };
    void dajGlos() {
        cout << "Rawr" << endl;
    }
};

int main()
{
    Rys r;
    Tygrys t;
    r.dajGlos();
    t.dajGlos();
}
```

Rys. 9.10. Kod języka programowania C++ – proste dziedziczenie (część 2)

```
Tworze Kotowaty
Tworze Rysia
Tworze Kotowaty
Tworze Tygrysa
Miauuu
Rawr
Usuwam Tygrysa
Usuwam Kotowaty
Usuwam Rysia
Usuwam Kotowaty

Process returned 0 (0x0)   execution time : 0.093 s
Press any key to continue.
```

Rys. 9.11. Wynik działania programu – przykład wykonania prostego dziedziczenia

Na rys. 9.11 widzimy, że najpierw uruchomił się konstruktor rodzica, a następnie konstruktor dziecka. Zawsze włączają się konstruktory od najbardziej bazowego. Na końcu działania programu, w odwrotnej kolejności, uruchomiły się odpowiednie destruktory. Ryś został utworzony pierwszy i usunięty przez program jako ostatni. C++ jest dość mądry w tej kwestii, bo nie chcemy sytuacji, w której zachodzą interakcje między rysiem i tygrysem, a ryś nagle znika i otrzymujemy informacje o błędach od kompilatora.

Klasy zaprzyjaźnione

Czasem chcemy dostać się do prywatnej bądź chronionej części klasy z innej klasy. Aby to zrobić, możemy utworzyć w klasie, z której chcemy pobrać informacje inne niż publiczne, deklarację w postaci:

```
friend class Nazwa_klasy;
```

W ten sposób tworzymy jednostronną „przyjaźń". Nasz przyjaciel ma dostęp do wszystkich zmiennych w klasie.

Na bazie naszej klasy kotowate utwórzmy nową klasę kot, która wygląda tak jak na rys. 9.12.

```
class Kot : public Kotowate {
private:
    string imie;
public:
    // Konstruktor
    Kot(string imie) :Kotowate() {
        cout << "Tworze kota" << endl;
        // Przypisanie wartości z parametru do zmiennej w klasie
        this->imie = imie;
    };
    Kot(const Kot& kot) :Kotowate(kot) { // Konstruktor kopiujący
        this->imie = kot.imie;
    };
    ~Kot() { cout << "Usuwam kota" << endl; };
    friend class Wlasciciel; // Deklaracja funkcji zaprzyjaźnionej
private:
    void imieKota() {
        cout << imie << endl;
    }
};
```

Rys. 9.12. Kod języka programowania C++ – klasy zaprzyjaźnione (kot)

Klasa właściciela kota będzie wyglądać tak jak na rys. 9.13.

```
class Wlasciciel {
public:
    Wlasciciel() {};
    // Przesyłamy obiekt kot z użyciem &, bez "&" przesłalibyśmy
    // tylko kopię kota i destruktor wywołałby się 2 razy
    void jakSieWabi(Kot& kot) {
        cout << "Kot ma na imie: ";
        kot.imieKota();
    }
};


int main()
{
    Kot kot1("Mruczek");
    Wlasciciel w1;
    w1.jakSieWabi(kot1);
}
```

Rys. 9.13. Kod języka programowania C++ – klasy zaprzyjaźnione (właściciel)

```
Tworze Kotowaty
Tworze kota
Kot ma na imie: Mruczek
Usuwam kota
Usuwam Kotowaty

Process returned 0 (0x0)   execution time : 0.090 s
Press any key to continue.
```

Rys. 9.14. Wynik działania kodu – zaprzyjaźnianie między klasami

Obiekt klasy właściciel ma dostęp do funkcji prywatnej obiektu klasy kot. Deklarację przyjaźni możemy zrobić w każdym miejscu klasy, poza metodami.

9.5. Składnik statyczny klasy i metody ich obsługi

Znasz już klasy i obiekty. Z poprzednich lekcji wiesz, że każdy obiekt jest pojedynczą instancją klasy, który ma własną zawartość. A co w przypadku, gdybyśmy chcieli przechowywać wspólne pole dla każdego kota? Przykładowo: liczba kotów. Musimy wtedy utworzyć **pole statyczne** i **metodę statyczną**. Takie pole będzie jedno dla klasy, ale będzie miał do niego dostęp każdy jej obiekt.

Aby utworzyć pole bądź metodę statyczną, w języku programowania C++ używamy słowa kluczowego „static".

```
class Kot : public Kotowate {
private:
    string imie;
public:
    // Konstruktor
    Kot(string imie) :Kotowate() {
        cout << "Tworze kota: " << imie << endl;
        this->imie = imie;
        liczKoty++;
    };
    Kot(const Kot& kot) :Kotowate(kot) { // Konstruktor kopiujący
        this->imie = kot.imie;
    };
    ~Kot() { cout << "Usuwam kota: "<< this->imie << endl; };
    // Deklaracja zmiennej statycznej
    static int liczKoty;

    static void liczbaKotow() {
        cout << "Aktualna liczba kotow: " << liczKoty << endl;
    }
};
```

Rys. 9.15. Kod języka programowania C++ – pola i metody statyczne w klasie (część 1)

```cpp
// Inicjalizacja wartości pola statycznego
int Kot::liczKoty = 0;

int main()
{
    Kot kot1("Mruczek");
    Kot::liczbaKotow();
    Kot kot2("Puszek");
    Kot::liczbaKotow();
    Kot kot3("Kulek");
    Kot::liczbaKotow();
}
```

Rys. 9.16. Kod języka programowania C++ – pola i metody statyczne w klasie (część 2)

```
Tworze Kotowaty
Tworze kota: Mruczek
Aktualna liczba kotow: 1
Tworze Kotowaty
Tworze kota: Puszek
Aktualna liczba kotow: 2
Tworze Kotowaty
Tworze kota: Kulek
Aktualna liczba kotow: 3
Usuwam kota: Kulek
Usuwam Kotowaty
Usuwam kota: Puszek
Usuwam Kotowaty
Usuwam kota: Mruczek
Usuwam Kotowaty

Process returned 0 (0x0)   execution time : 0.028 s
Press any key to continue.
```

Rys. 9.17. Wynik działania kodu – pola i metody statyczne w klasie

Warto pamiętać, że:

- każda zmienna statyczna powstaje jedna dla wszystkich obiektów;
- zmiennych statycznych nie można inicjalizować wewnątrz ciała klasy;
- zmienne statyczne są zapisywane w innej części pamięci komputera niż zwykłe zmienne;
- zmienne są tworzone na początku programu i usuwane na samym końcu;
- zmienne statyczne mogą modyfikować wyłącznie metody / funkcje statyczne.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień i opisz kwalifikatory dostępu w języku programowania C++.
2. Scharakteryzuj konstruktory i destruktory. Wyjaśnij, do czego służą w programowaniu obiektowym.
3. Wyjaśnij pojęcia klas dziedziczonych i klas zaprzyjaźnionych.
4. Opisz zagadnienie: składnik statyczny klasy i metody jego obsługi.
5. Stwórz klasę statyczną i metodę statyczną w języku programowania C++.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1.
   A. Stwórz plik klasa.cpp, a w nim klasę w języku programowania C++.
   B. W klasie stworzonej w pliku klasa.cpp umieść trzy właściwości i co najmniej jedną dowolną metodę.
   C. Utwórz plik klasa.py i wykonaj w nim za pomocą języka programowania Python polecenia tożsame do opisanych w punktach A i B.
2. Na bazie klas zakodowanych w zadaniu 1. utwórz kilka przykładowych obiektów w językach programowania:
   A. C++,
   B. Python 3.x.

10 Klasy pochodne i wzorce klas

ZAGADNIENIA

- Hierarchia dziedziczenia klas w programie
- Klasy bazowe i pochodne
- Metody wirtualne i abstrakcyjne klasy

10.1. Hierarchia dziedziczenia klas w programie

Wiemy już, jak tworzyć własne klasy i dziedziczenie między klasami. Teraz chcemy stworzyć więcej dziedziczeń. Jak zachowają się wtedy konstruktory? Dla przykładu utworzymy trzy klasy: A, B i C, gdzie A będzie rodzicem B, a B będzie rodzicem C, aby przeanalizować kolejność wywoływania destruktorów i konstruktorów.

```
#include <iostream>
using namespace std;

class A {
public:
    // Konstruktor bezparametrowy
    A() { cout << "Tworze a" << endl; };
    // Konstruktor kopiujący
    A(const A& a) {};
    //Destruktor
    ~A() { cout << "Usuwam a" << endl; };
};
```

Rys. 10.1. Kod języka programowania C++ – hierarchia dziedziczenia (część 1)

```
class B : public A{
public:
    // Konstruktor bezparametrowy
    B():A() { cout << "Tworze b" << endl; };
    // Konstruktor kopiujący
    B(const B& b):A(b) {};
    //Destruktor
    ~B() { cout << "Usuwam b" << endl; };
};
```

Rys. 10.1. Cd. Kod języka programowania C++ – hierarchia dziedziczenia (część 1)

```
class C : public B{
public:
    // Konstruktor bezparametrowy
    C():B() { cout << "Tworze c" << endl; };
    // Konstruktor kopiujący
    C(const C& c):B(c) {};
    // Destruktor
    ~C() { cout << "Usuwam c" << endl; };
};


int main()
{
    C c;
}
```

Rys. 10.2. Kod języka programowania C++ – hierarchia dziedziczenia (część 2)

```
Tworze a
Tworze b
Tworze c
Usuwam c
Usuwam b
Usuwam a

Process returned 0 (0x0)   execution time : 0.092 s
Press any key to continue.
```

Rys. 10.3. Wynik działania kodu – hierarchia dziedziczenia

Jako pierwszy zawsze uruchamia się konstruktor najbardziej podstawowego rodzica, na bazie którego tworzymy klasę, czyli klasę najbardziej bazową w hierarchii.

Pod koniec programu destruktory włączają się zawsze w odwrotnej kolejności: zaczynają od klas pochodnych i kończą na klasie bazowej.

10.2. Klasy bazowe i pochodne

Największą zaletą dziedziczenia w programowaniu jest możliwość utworzenia nowych obiektów dzieci na wskaźnik do rodzica. Umożliwia to umieszczenie w tablicy wskaźnikowej typu rodzica obiektów wszystkich jego dzieci. Aby pokazać to na prostszym przykładzie, użyjemy zwykłego wskaźnika na rodzica, a potem utworzymy na nim instancję dziecka, pozbędziemy się go i utworzymy kolejne. We wcześniejszym kodzie musimy zamienić destruktory klas, z których dziedziczymy, na destruktory wirtualne z użyciem słówka kluczowego „virtual". **Zapamiętaj, że zawsze podczas pierwszego dziedziczenia i pracy na typach wskaźnikowych trzeba używać w klasach rodziców wirtualnego destruktora.**

```cpp
#include <iostream>
using namespace std;

class A {
public:
    // Konstruktor bezparametrowy
    A() { cout << "Tworze a" << endl; };
    // Konstruktor kopiujący
    A(const A& a) {};
    //Destruktor
    virtual ~A() { cout << "Usuwam a" << endl; };
};
```

Rys. 10.4. Kod języka programowania C++ – klasy bazowe i pochodne (część 1)

```
class B : public A{
public:
    // Konstruktor bezparametrowy
    B():A() { cout << "Tworze b" << endl; };
    // Konstruktor kopiujący
    B(const B& b):A(b) {};
    //Destruktor
    virtual ~B() { cout << "Usuwam b" << endl; };
};
```

Rys. 10.4. Cd. Kod języka programowania C++ – klasy bazowe i pochodne (część 1)

```
class C : public B{
public:
    // Konstruktor bezparametrowy
    C():B() { cout << "Tworze c" << endl; };
    // Konstruktor kopiujący
    C(const C& c):B(c) {};
    // Destruktor
    ~C() { cout << "Usuwam c" << endl; };
};


int main()
{
    A* a;
    a = new C();
    delete a;
    cout << endl;
    a = new B();
    delete a;
}
```

Rys. 10.5. Kod języka programowania C++ – klasy bazowe i pochodne (część 2)

```
Tworze a
Tworze b
Tworze c
Usuwam c
Usuwam b
Usuwam a

Tworze a
Tworze b
Usuwam b
Usuwam a

Process returned 0 (0x0)   execution time : 0.091 s
Press any key to continue.
```

Rys. 10.6. Wynik działania programu – klasy bazowe i pochodne

Spójrz na kod w funkcji **`main`**. Mając jeden wskaźnik na klasę, możemy obsłużyć wszystkie pochodne tej klasy. Daje nam to nieskończone możliwości, np. jesteśmy w stanie napisać funkcję, do której będzie podawać „dzieci" pewnej klasy i robić na nich masowe operacje, zamiast pisać wiele funkcji dla każdej pochodnej klasy.

10.3. Metody wirtualne i abstrakcyjne klasy

W świecie rzeczywistym nie ma zwierząt „kotowate", ale jest rodzina kotowatych. W programowaniu obiektowym też możemy tworzyć klasy abstrakcyjne – bez ich obiektów, natomiast z kilkoma ciekawymi własnościami.

Na rys. 10.7 powinieneś zauważyć, że:

- metoda „daj glos()" w klasie „Kotowate" ma etykietę „virtual" i jest przypisana do 0 – to czyni z niej metodę wirtualną;
- wystarczy jedna wirtualna metoda, aby klasa zamieniła się w klasę abstrakcyjną – zabiera nam to możliwość tworzenia obiektów tej typu klasy, ale zezwala na bardziej zaawansowane zagadnienia polimorfizmu;
- metoda „dajGlos()" w rodzicu zwraca typ „void" – odpowiadająca jej metoda we wszystkich dzieciach, które mają być nieabstrakcyjne, musi mieć metodę o nazwie „dajGlos()", która zwraca typ „void" (typ niezwracający niczego) – tylko taka metoda zmieni klasę dziecka w klasę nieabstrakcyjną; analogicznie trzeba tak postąpić z każdą wirtualną metodą, jaką będzie nasza klasa dziedziczyć.

```cpp
#include <iostream>
using namespace std;

class Kotowate {
public:
    //Konstruktor bezparametrowy
    Kotowate() { cout << "Tworze Kotowaty" << endl; };
    //Konstruktor kopiujący
    Kotowate(const Kotowate& kotowate) {};
    //Destruktor
    ~Kotowate() { cout << "Usuwam Kotowaty" << endl; };
    virtual void dajGlos() = 0;
};

class Rys : Kotowate {
public:
    Rys() :Kotowate() { cout << "Tworze Rysia" << endl; };
    Rys(const Rys& rys) :Kotowate(rys) {};
    ~Rys() { cout << "Usuwam Rysia" << endl; };
    void dajGlos() {
        cout << "Miauuu" << endl;
    }
};
```

Rys. 10.7. Kod języka programowania C++ – przykład dziedziczenia klasy ryś po klasie kotowate

Obserwując kod klasy „Rys" (rys. 10.7) i klasy „Tygrys" (rys. 10.8), trzeba pamiętać, że podczas dziedziczenia:

- każdy konstruktor klasy dziecka musi wywołać odpowiadający mu konstruktor klasy rodzica;
- destruktora nigdzie nie wywołujemy – robi to kompilator; zawsze musi istnieć destruktor, nawet jeżeli miałby być pusty.

CIEKAWOSTKA

Podczas dziedziczenia po wielu różnych klasach konstruktor dziecka wywołuje konstruktory rodziców w postaci:

```
class A: B,C{
public:
A():B(),C(){…};
…
};
```

```
class Tygrys : Kotowate {
public:
    Tygrys() :Kotowate() { cout << "Tworze Tygrysa" << endl; };
    Tygrys(const Tygrys& tygrys) :Kotowate(tygrys) {};
    ~Tygrys() { cout << "Usuwam Tygrysa" << endl; };
    void dajGlos() {
        cout << "Rawr" << endl;
    }
};

int main()
{
    Rys r;
    Tygrys t;
    r.dajGlos();
    t.dajGlos();
}
```

Rys. 10.8. Kod języka programowania Python – przykład dziedziczenia klasy tygrys po klasie kotowate

Metoda **"dajGłos()"** została zapisana w dość niecodzienny sposób. Zmienia ona naszą klasę kotowate w klasę abstrakcyjną (nie możemy utworzyć jej obiektu). Klasy dziedziczące po niej muszą natomiast nadpisać w sobie wszystkie metody zapisane w taki sposób. Nazywamy to **metodami wirtualnymi**. Bez takiej metody wirtualnej nie moglibyśmy na przykład dodać obiektów klas ryś i tygrys do jednej tablicy typu kotowate, a potem iterować po elementach (obiektach różnych klas o wspólnej klasie bazowej) wywołanie metody **"dajGłos()"**. Byłoby to niemożliwe, a taki zapis upraszcza wiele rzeczy.

```
Tworze Kotowaty
Tworze Rysia
Tworze Kotowaty
Tworze Tygrysa
Miauuu
Rawr
Usuwam Tygrysa
Usuwam Kotowaty
Usuwam Rysia
Usuwam Kotowaty

Process returned 0 (0x0)   execution time : 0.101 s
Press any key to continue.
```

Rys. 10.9. Wynik działania programu – dziedziczenie między klasą bazową a jej pochodnymi

Co by było, gdybyśmy nie zapisali metody **"dajGłos()"** w klasie bazowej ze słówkiem virtual, a w jej wnętrzu zapisali **"cout << "cokolwiek" << endl;"**? W takim przypadku metody z klas pochodnych zasłoniłaby metoda z klasy rodzica, więc zamiast **"Miauuu"** i **"Rawr"** dostalibyśmy dwa razy **"cokolwiek"**.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Do czego służy słówko kluczowe „virtual” w C++?
2. Opisz, w jakiej kolejności wywołują się konstruktory.
3. Opisz, w jakiej kolejności wywołują się destruktory.
4. Co trzeba zrobić, aby dziedziczyć po klasie wirtualnej?

11 Programowanie wyjątków

ZAGADNIENIA

- Instrukcje `try` i `catch`
- Instrukcja `throw`
- Obsługa błędów wykonania aplikacji

11.1. Instrukcje `try` i `catch`

W językach programowania możemy przewidzieć różne zdarzenia i je zabezpieczyć. Niestety, nie jesteśmy w stanie dokładnie przewidzieć np. zachowania użytkownika bądź czy na pewno istnieje określona baza danych. To dlatego wymyślono instrukcje sprawdzające, które zabezpieczają wystąpienie potencjalnych błędów pamięci i ewentualnie przerywających program, zanim dojdzie do nieoczekiwanych błędów, czego początkujący programiści często nadużywają.

Instrukcja w C++ składa się z bloku **`try`**, który próbuje wykonać część kodu, i z bloku **`catch`**, który przechwytuje błędy i zwraca komunikat, zabezpieczając program przed niekontrolowanym wyciekiem danych.

Kod w uproszczeniu wygląda następująco:

```
try{
    Instrukcje;
}catch(typ nazwa){
    Instrukcje;
}catch(...){
    Instrukcje;
}
```

Catch jako parametr może przyjmować typ błędu lub „...” – wtedy przechwytuje wszystkie możliwe błędy, zakładając, że nie przechwycił go inny blok catch. Po jednym bloku try możemy umieścić nie mniej niż jeden blok przechwytujący catch – nie ma górnego limitu przechwyconych błędów.

11.2. Instrukcja `throw`

Aby przekazać konkretny błąd bądź numer błędu do bloku catch, używamy słowa kluczowego „throw". Może on zwrócić różne typy. Na rys. 11.1 zwraca on inta (numerycznego) o przykładowej wartości 101. Wartości błędów to tylko lokalny sposób ujednolicenia oprogramowania, więc warto mieć swój system numeracji błędów, gdy pracuje się przy większych ilościach kodu.

```
#include <iostream>
using namespace std;

float dzielenie(float a, float b) {
    if (b == 0) {
        throw 101;
    }
    else {
        return a / b;
    }
}

int main()
{
    float a = 3;
    float b = 0;
    float wynik_dzielenia;
    try {
        wynik_dzielenia = dzielenie(a, b);
    }
    catch (int kod_bledu) {
        cout << "Wystapil blad: "<< kod_bledu << endl;
        if (kod_bledu == 101) {
            cout << "Nie mozna dzielic przez zero!" << endl;
        }
    }
}
```

Rys. 11.1. Kod języka programowania C++ – przykład **try-throw-catch**

```
Wystapil blad: 101
Nie mozna dzielic przez zero!

Process returned 0 (0x0)   execution time : 0.096 s
Press any key to continue.
```

Rys. 11.2. Wynik działania programu – przykład **try-catch-throw**

Musimy zawsze pamiętać o dwóch rzeczach:

- umieszczać blok „catch” po bloku „try”;
- umieszczać instrukcję „throw” wyłącznie w bloku wywołania „try” (dla powyższego przykładu funkcji „dzielenie” nie możemy wywołać poza blokiem „try”, próba wywołania funkcji „dzielenie” poza blokiem „try” skutkuje błędem).

11.3. Obsługa błędów wykonania aplikacji

Na poniższych przykładach pokazano przechwytywanie błędów od użytkownika i wymuszanie poprawnego wykorzystania programu przez użytkownika.

```cpp
#include <iostream>
using namespace std;

int main()
{
    // deklaracja zmiennych
    float a = 3;
    float b = 0;
    float wynik_dodawania;
    // włączenie przechwytywania błędów
    strumienia wejściowego
    cin.exceptions(std::ios_base::failbit);
    while(true){
        cout << "Podaj pierwsza liczbe:" << endl;
        try {
            cin >> a; // próba pochwycenia floata
            break; // jeżeli próba się powiedzie,
            to while zostanie przerwane
        }
```

Rys. 11.3. Kod języka programowania C++ – przykład **try-catch** na przykładzie obsługi błędów wejścia od użytkownika (część 1)

```
    catch (std::ios_base::failure const& ex)
    {
        cout << "Nie podales liczby!!" <<
        endl;
        // formuła na czyszczenie "śmieci"
        na obiekcie cin
        std::cin.clear(); //czyszczenie flagi
        // czyszczenie ewentualnych znaków,
        które mogłyby występować po
        // białych znakach
        std::cin.ignore
        (std::numeric_limits<std::streamsize>
        ::max(), '\n');
    }
}
```

Rys. 11.4. Kod języka programowania C++ – przykład **try-catch** na przykładzie obsługi błędów wejścia od użytkownika (część 2)

```
        catch (std::ios_base::failure
        const& ex) {
            cout << "Nie podales liczby!!"
            << endl;
            //czyszczenie strumienia
            wejścia, potrzebujemy tego,
            gdyby użytkownik podał
            jakiekolwiek białe znaki
            std::cin.clear();
            std::cin.ignore
            (std::numeric_limits<std::strea
            msize>::max(), '\n');
        }
    }
    //dalsza część programu - nie pojawi
    się żaden błąd,bo try i catch zmusił
    użytkownika do podania liczb
    wynik_dodawania = a + b;
    cout << a << " + " << b << " = " <<
    wynik_dodawania << endl;
}
```

Rys. 11.5. Kod języka programowania C++ – przykład **try-catch** na przykładzie obsługi błędów wejścia od użytkownika (część 3)

```
//Pętla, która będzie się tak długo
wykonywać, aż użytkownik nie poda
odpowiedniego wejścia - drugiej liczby
while (true) {
    cout << "Podaj druga liczbe:" <<
    endl;
    //próba pobrania liczby ze
    strumienia
    try {
        //gdy użytkownik wpisze liczbę
        - uruchomi się "break"
        i przerwie pętlę while
        cin >> b;
        break;
    }
    //w przeciwnym wypadku zostanie
    utuchomiony "catch" i wyczyści
    strumień, zaczynając pętlę
    od początku
```

Rys 11.6. Kod języka programowania C++ – przykład **try-catch** na przykładzie obsługi błędów wejścia od użytkownika (część 4)

```
Podaj pierwsza liczbe:
jeden
Nie podales liczby!!
Podaj pierwsza liczbe:
1
Podaj druga liczbe:
dw a
Nie podales liczby!!
Podaj druga liczbe:
2
1 + 2 = 3

Process returned 0 (0x0)   execution time : 0.091 s
Press any key to continue.
```

Rys. 11.7. Wynik działania programu – przykład **try-catch** na przykładzie obsługi błędów wejścia od użytkownika

SPRAWDŹ SWOJĄ WIEDZĘ

1. Przekształć zabezpieczenie podania liczby przez użytkownika z rozdziału 11.3 w funkcję.
2. Wykorzystaj funkcję z zadania 1. i utwórz prosty kalkulator w konsoli.
3. Jaka jest minimalna liczba bloków „catch", której można użyć dla pojedynczego bloku „try"?
4. Opisz kolejność bloków i metody przechwytywania błędów w języku programowania C++.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Stwórz plik test.cpp. Napisz program konsolowy, który będzie pobierał dwie dane od użytkownika. Jeżeli jedna z podanych danych nie była liczbą całkowitą, program musi poinformować o tym użytkownika i czekać na podanie właściwej liczby. Jeśli użytkownik wprowadzi poprawnie dwie liczby całkowite, powinien otrzymać informację, np. „Brawo! Oba wpisy to liczby całkowite".

III. Programowanie aplikacji desktopowych

- **Środowisko programistyczne dla aplikacji desktopowych**
- **Wykorzystywanie frameworków do programowania aplikacji desktopowych**
- **Programowanie desktopowych aplikacji okienkowych za pomocą języka programowania Python**

12 Środowisko programistyczne dla aplikacji desktopowych

ZAGADNIENIA
- Środowiska IDE
- Środowiska RAD
- Framework

12.1. Środowisko IDE

Zintegrowane środowisko programistyczne, czyli IDE (ang. *Integrated Development Environment*), to program umożliwiający pisanie, modyfikowanie i debugowanie programów. Jego główne funkcjonalności to:

- podkreślanie błędów, ostrzeżeń i uwag dotyczących standaryzacji kodu;
- automatyczne podpowiadanie nazw kluczowych języka, nazw zmiennych, funkcji, klas itp.;
- refactoring kodu – wprowadzanie wybranej standaryzacji;
- kontrolowanie wersji produktu / aplikacji;
- pomoc w debugowaniu kodu;
- przeszukiwanie kodu;
- linkowanie dodatkowych bibliotek;
- kompilowanie kodu na różne platformy.

Przykłady IDE:

- Microsoft Visual Studio – zezwala na pisanie aplikacji niemal w każdym języku, działa tylko na systemach operacyjnych Windows.
  Oficjalna strona projektu – https://visualstudio.microsoft.com/pl/
- Eclipse – pozwala na tworzenie aplikacji w języku programowania Java, jest całkowicie otwarte.
  Oficjalna strona projektu – https://www.eclipse.org/
- IntelliJ – domyślnie pozwala na programowanie w języku Java, możliwe jest programowanie w nim w innych językach po instalacji odpowiednich wtyczek.
  Oficjalna strona projektu – https://www.jetbrains.com/idea/
- Code::Blocks – umożliwia programowanie i pisanie aplikacji w językach programowania C i C++, jest darmowe i otwarte.
  Oficjalna strona projektu – https://www.codeblocks.org/downloads/

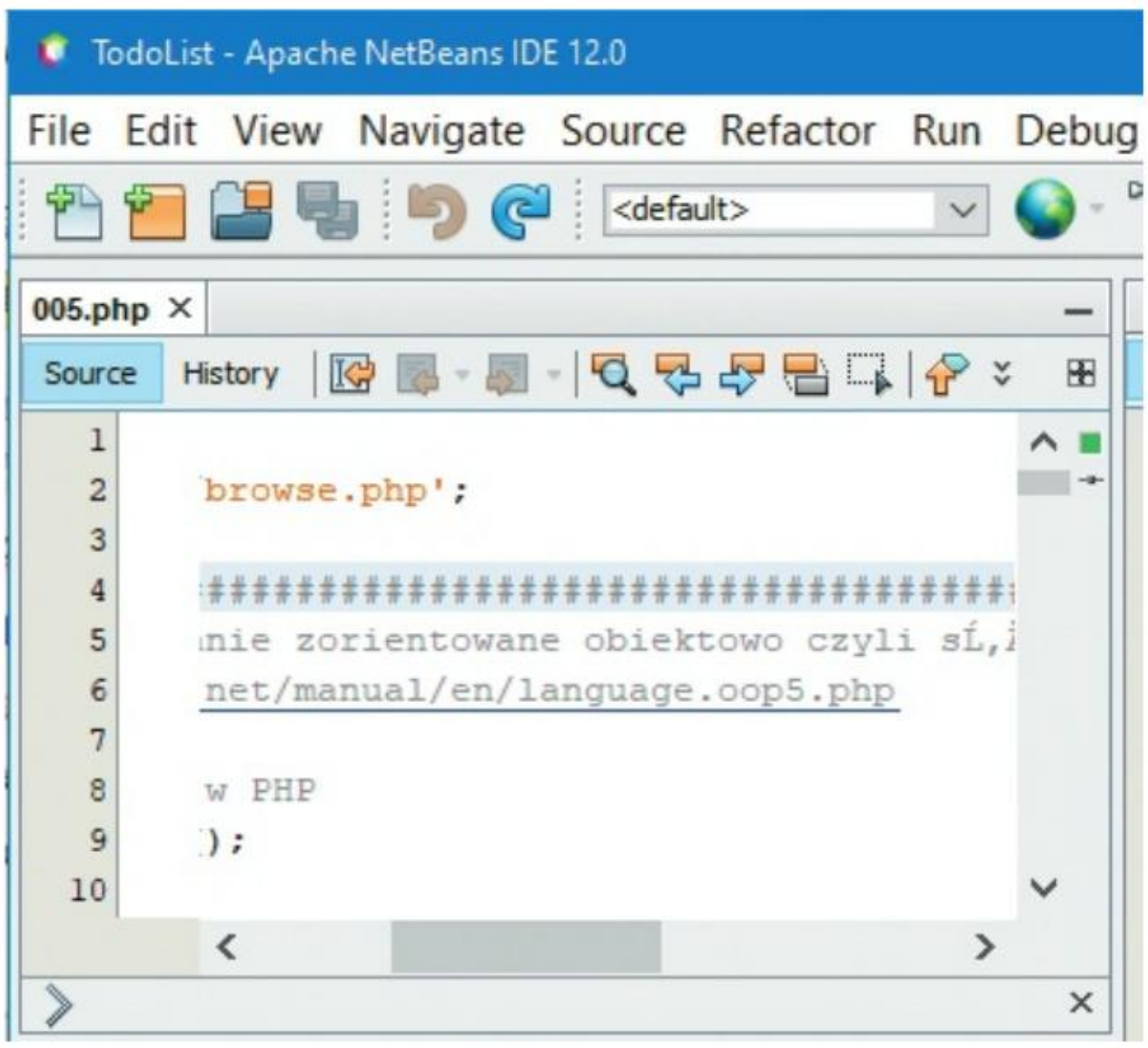
- Visual Studio Code – darmowe, dostępne na Windows, iOS i Linux, rozwijane przez ludzi z całego świata, można w nim programować najprawdopodobniej w każdym istniejącym języku programowania, wystarczy znaleźć odpowiednią wtyczkę. Oficjalna strona projektu – https://code.visualstudio.com/
- Apache NetBeans – bezpłatne, rozwijane przez The Apache Software Foundation (strona instytucji: https://apache.org/), dostępne na Windows, Linux, Mac OSX i BSD. Oficjalna strona projektu – https://netbeans.apache.org/

CIEKAWOSTKA

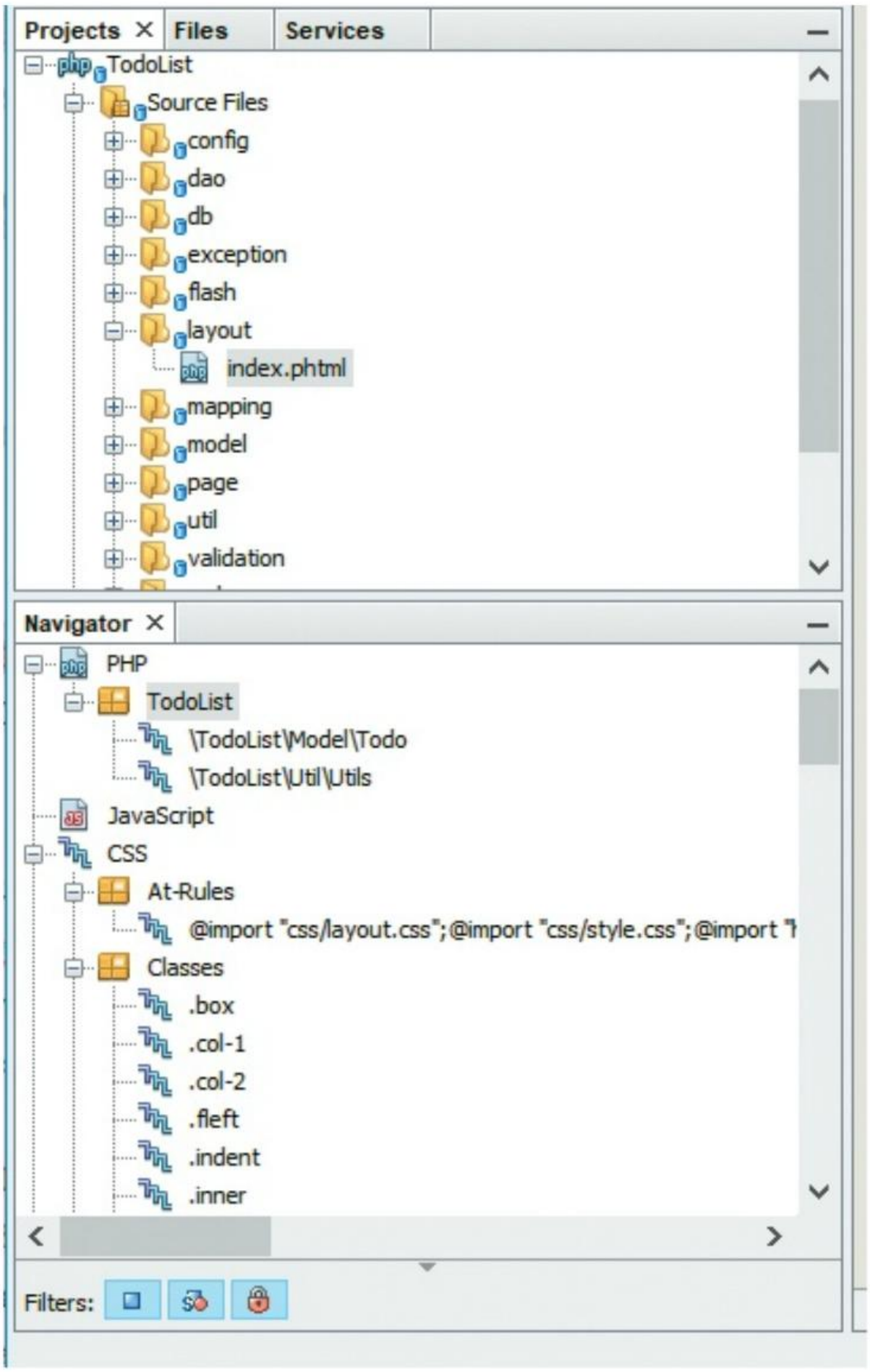
The Apache Software Foundation jest w obszarze oprogramowania jedną z największych na świecie organizacji non profit. Dba ona o rozwój ponad 350 projektów informatycznych. Na stronie https://apache.org/index.html#projects-list znajdziesz dostęp do każdego z nich. Jedne z najbardziej znanych aplikacji rozwijanych przez tę organizację to:

- pakiet XAMPP,
- Apache NetBeans (panel aplikacji przedstawiono na rys. 12.1),
- Apache Cassandra,
- Apache Tomcat,
- Apache Doris,
- Apache Cordova.

Dzięki tego typu organizacjom ludzie na całym świecie mają dostęp do nowoczesnego i bezpłatnego rozwiązania.



Rys. 12.1. Widok lewego panelu aplikacji Apache NetBeans



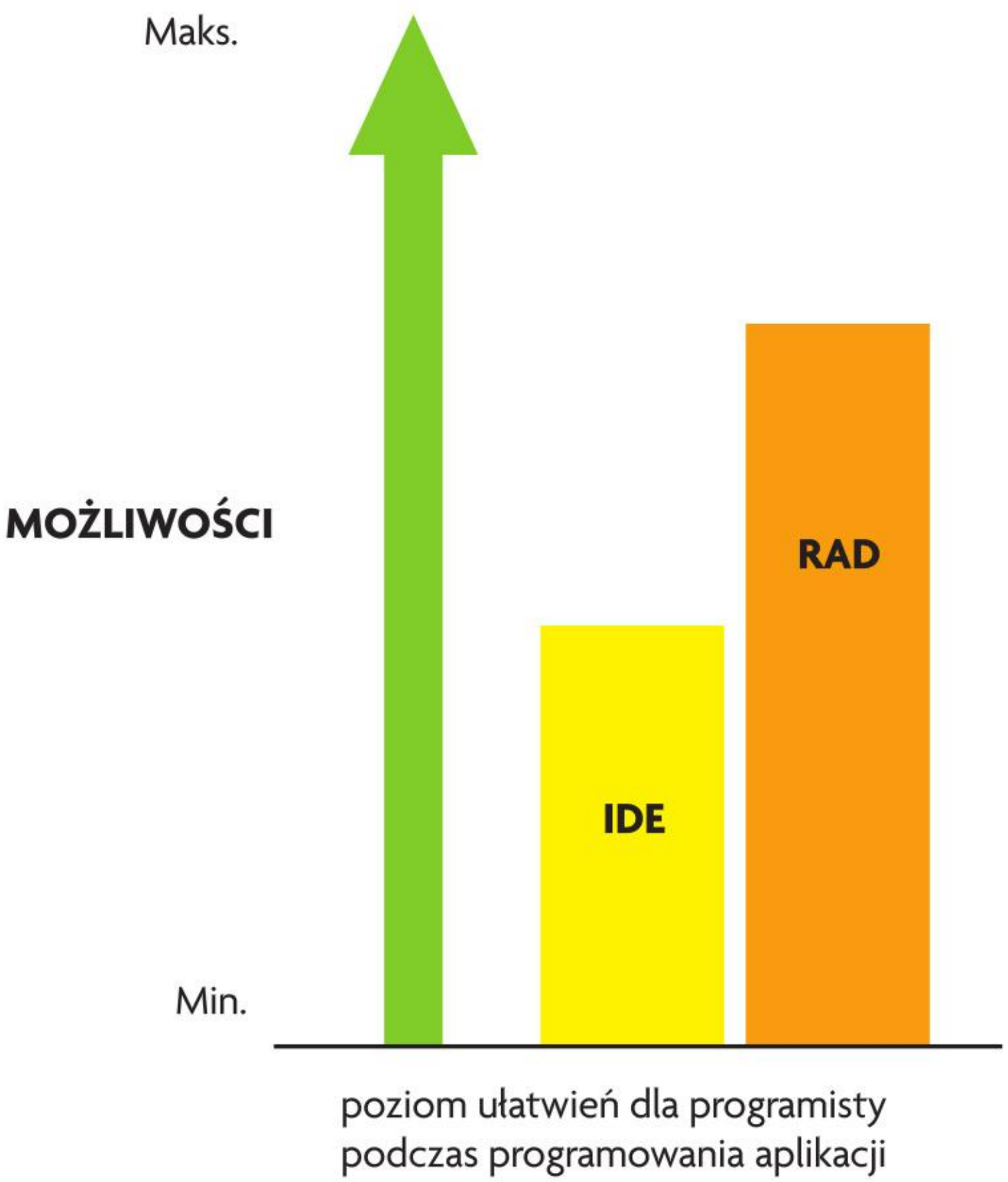
Rys. 12.1. Cd. Widok lewego panelu aplikacji Apache NetBeans

12.2. Środowisko RAD

Rapid Application Development (RAD) to rozszerzone IDE. Poza pomocą w programowaniu dostarcza narzędzi przyspieszających budowę interfejsów graficznych dla użytkowników. Umożliwia pisanie widoków graficznych w prosty, szybki i wygodny sposób. Co więcej, takie środowisko wspomaga programistów w pracy z bazami danych.

Przykłady środowisk RAD:

- Microsoft Visual Studio – płatne rozwiązanie do tworzenia aplikacji okienkowych w C#, ASP.net Core, ASP.net Framework.
  Oficjalna strona projektu – https://visualstudio.microsoft.com/pl/
- RAD Studio – płatne rozwiązanie do tworzenia aplikacji okienkowych w językach programowania C++ i Delphi.
  Oficjalna strona projektu – https://www.embarcadero.com/products/rad-studio
- Qt Designer – płatne rozwiązanie do tworzenia aplikacji okienkowych w językach programowania C++, Python, QML i Java.
  Oficjalna strona projektu – https://www.qt.io/download



Rys. 12.2. IDE a RAD – poziom wsparcia dla programisty w pisaniu kodu

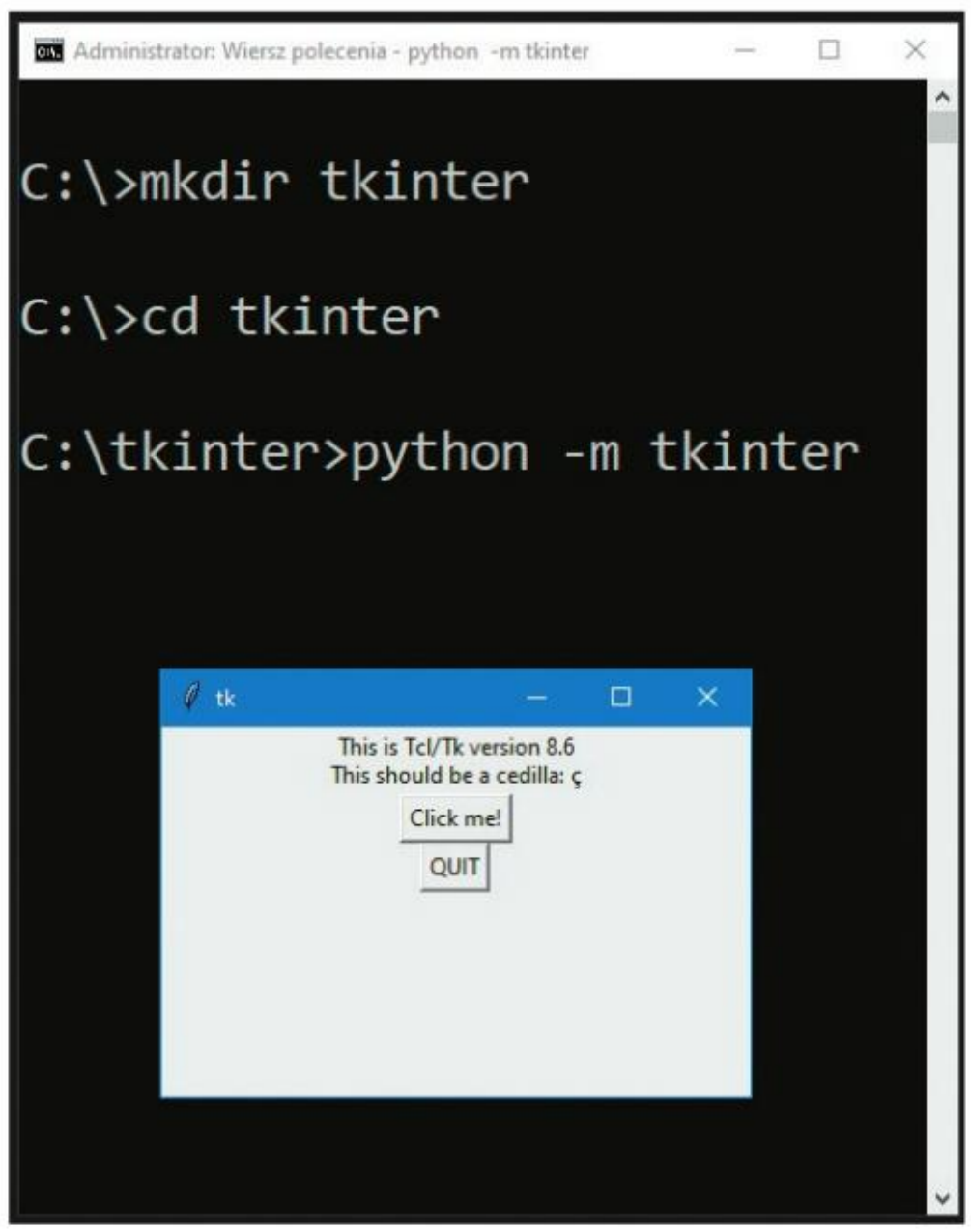
12.3. Framework

Jest to szkielet aplikacji zawierający zestaw bibliotek graficznych, dostarczający programiście system zdarzeń i gotowe kontrolki (biblioteki i komponenty graficzne) umożliwiające tworzenie w prosty sposób zaawansowanych aplikacji z interfejsem graficznym użytkownika.

Frameworki często nazywa się bibliotekami. Nie jest to wielki błąd, lecz możliwości frameworków są bardziej rozwinięte.

Przykładowe frameworki:

- Framework QT – pozwala na tworzenie wieloplatformowych aplikacji okienkowych z graficznym interfejsem użytkownika.
  Oficjalna strona projektu – https://www.qt.io/product/framework
- Tkinter – framework Pythona pozwalający na budowę aplikacji okienkowych jest darmowy, otwarty i stosunkowo prosty w użyciu.
  Oficjalna strona projektu – https://docs.python.org/3/library/tkinter.html
- WPF (*Windows Presentation Foundation*) – pozwala na tworzenie w szybki i prosty sposób za pomocą języka znaczników XAML interfejsów graficznych użytkowników.
  Oficjalna strona projektu – https://docs.microsoft.com/pl-pl/dotnet/desktop/wpf/?view=netdesktop-5.0



Rys. 12.3. Polecenie **`python -m tkinter`** – sprawdza poprawność instalacji i wersję frameworku tkinter w systemie operacyjnym

CIEKAWOSTKA

Po wpisaniu w konsoli systemowej polecenia:

```
python -m tkinter
```

wyskoczy okienko informacyjne z wersją frameworku tkinter (rys. 12.3). Aby tak się stało, tkinter musi być poprawnie zainstalowany w systemie operacyjnym.

Zalety frameworków:

- rozwiązania są przetestowane i sprawnie funkcjonują;
- narzucają pewną logikę pisania kodu, przez co łatwiej tworzyć projekty przy dużej liczbie programistów;
- skracają czas pisania aplikacji – programiści nie muszą się martwić, jak powinien działać przycisk, bo wystarczy go tylko zaimportować i wstawić w odpowiednie miejsce aplikacji.

Wady frameworków:

- narzuconej logiki frameworków trzeba się nauczyć i ściśle się jej trzymać, aby aplikacja działała optymalnie;
- frameworki są pisane tak, aby zaspokoić większość potrzeb programistów; niestety, oznacza to, że nie zawsze sposób ich działania jest najszybszy z możliwych;
- dodanie nowych funkcjonalności zajmuje dużo czasu i często wymaga długiej lektury dokumentacji, a w skrajnych przypadkach nawet kodów źródłowych frameworków.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co to jest środowisko IDE?
2. Wymień podstawowe funkcjonalności środowiska IDE.
3. Wymień trzy przykładowe IDE.
4. Co to jest środowisko RAD?
5. Co umożliwiają frameworki?
6. Wymień zalety i wady używania frameworków.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Sprawdź na swoim komputerze, jakie masz zainstalowane programy IDE. Porównaj je pod względem technologii, do których najlepiej się nadają.
   Jeżeli nie znajdziesz żadnego tego typu programu, poszukaj co najmniej dwóch przykładowych w internecie i je porównaj.
2. Sprawdź na swoim komputerze, jakie masz zainstalowane programy RAD. Porównaj je pod względem technologii, do których najlepiej się nadają.
   Jeżeli nie znajdziesz żadnego tego typu programu, poszukaj co najmniej dwóch przykładowych w internecie i je porównaj.

13 Wykorzystywanie frameworków do programowania aplikacji desktopowych

ZAGADNIENIA

- Elementy interfejsu użytkownika
- Okno aplikacji desktopowej
- Tekst
- Przyciski
- Wejście użytkownika
- Zaawansowane wejście użytkownika, obsługa klawiatury i zdarzenia myszy

13.1. Elementy interfejsu użytkownika

W poprzednich rozdziałach programowaliśmy aplikacje konsolowe, które można kontrolować tylko za pomocą klawiatury. W ich przypadku jesteśmy uzależnieni od położenia znaku zachęty w konsoli. Teraz zajmiemy się tworzeniem **graficznych interfejsów użytkownika** (GUI, ang. *graphical user interface*).

Wyróżnia się następujące elementy graficznych interfejsów:

- przyciski,
- tekst,
- okna modalne,
- okna niemodalne,
- wejście dla użytkownika,
- menu aplikacji,
- przyciski w menu aplikacji.

Oczywiście elementy graficzne nie zamykają tematu funkcjonalności aplikacji z graficznym interfejsem użytkownika. Programy mają różne systemy zdarzeń klawiatury (wciśnięcie przycisku / kombinacji przycisków) i myszy (najechanie na element, użycie prawego przycisku myszy bądź lewego). Większość tych zdarzeń i elementów wizualnych zapewniają frameworki. W następnych rozdziałach pokażemy wymienione elementy za pomocą frameworka tkinter z użyciem programowania obiektowego w języku programowania Python.

Aby zainstalować framework tkinter, można użyć polecenia w konsoli CMD, Powershellu lub terminalu Linuksa:

```
pip install tkinter
```

Jeżeli w systemie operacyjnym masz kilka wersji Pythona, warto zobaczyć, czy pip odwołuje się do Pythona 3. Aby to sprawdzić, wpisz w linię poleceń:

```
pip --version
```

Jeżeli nie zwróci to twojej wersji Pythona (najlepiej skorzystać z wersji 3.x.x), to użyj poniższego polecenia, aby zainstalować tkintera na Pytonie 3:

```
pip3 install tkinter
```

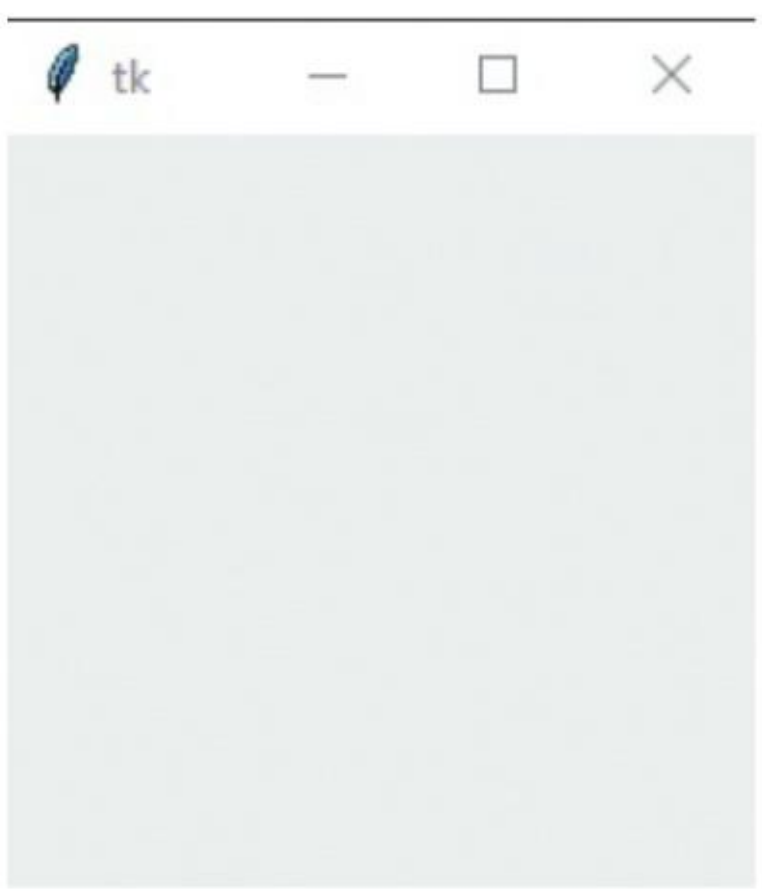
13.2. Okno aplikacji desktopowej

Aby sprawdzić, czy framework tkinter działa, spróbuj wpisać kod przedstawiony na rys. 13.1.

```
import tkinter

okno = tk.Tk()
okno.mainloop()
```

Rys. 13.1. Kod języka programowania Python – test działania frameworku tkinter



Rys. 13.2. Wynik działania programu – test działania frameworku tkinter

Jak widać na rys. 13.2, trzy proste linijki utworzyły aplikację okienkową, która używa bibliotek systemowych. Mamy działające przyciski wyłączenia i minimalizacji okna. Co więcej, możemy zmieniać wielkość okna.

Na rys. 13.1 przedstawiono kod, który:

- **`import tkinter as tk`** – importuje framework tkinter i tworzy alias do nazwy frameworku jako tk;
- **`okno = tk.Tk()`** – inicjalizuje okno tkintera za pomocą konstruktora okna;
- **`okno.mainloop()`** – włącza okno aplikacji i staruje pętlę programu.

Teraz możemy nadać własny tytuł oknu naszej aplikacji za pomocą metody **`title()`**:

```
okno.title("Nazwa aplikacji")
```

W taki sam sposób, za pomocą metody **`geometry`**, można dodać wymiary aplikacji:

```
okno.geometry("300×400")
```

Za pomocą metody **`resizable(False, False)`** można zablokować skalowanie okna aplikacji przez użytkownika.

Skalowanie aplikacji – blokada:

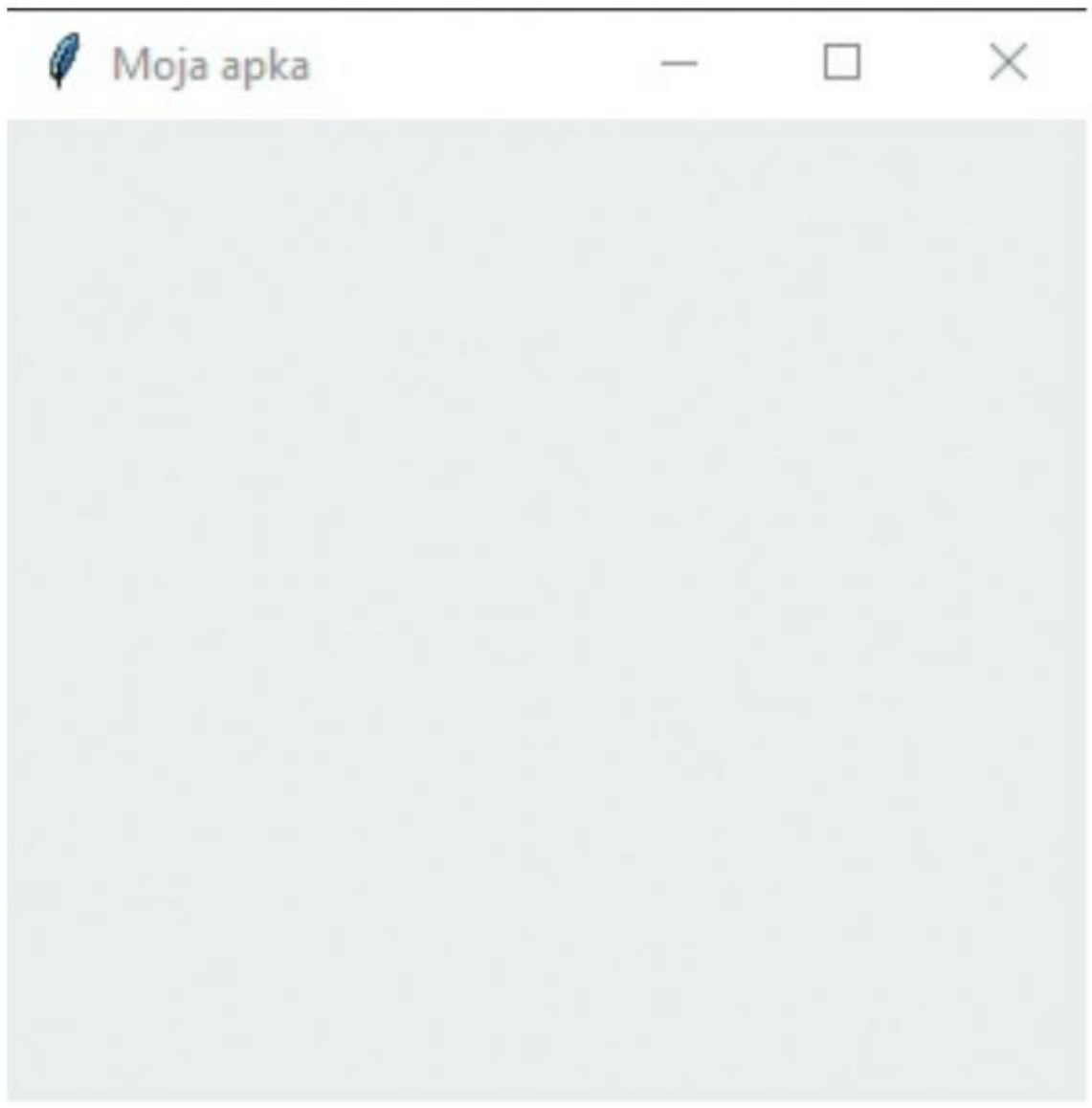
- pierwszy argument – na osi X,
- drugi argument – na osi Y.

Kod naszej aplikacji przedstawiono na rys. 13.3.

```
import tkinter

okno = tkinter.Tk()
# Dodanie nazwy okna
okno.title("Moja apka")
# dodanie wymiarów okna
okno.geometry("300x400")
# zablokowanie skalowania okna
okno.resizable(False, False)
okno.mainloop()
```

Rys. 13.3. Kod języka programowania Python – podstawy pisania okien we frameworku tkinter



Rys. 13.4. Wynik działania programu – podstawy pisania okien we frameworku tkinter

Jak widać na rys. 13.4, nie można już zmienić szerokości ani wysokości okna aplikacji. Pojawił się również jej tytuł.

13.3. Tekst

Do okna aplikacji okienkowej dodamy teraz tekst. Użyjemy do tego celu obiektu **`Label()`** z frameworku tkinter.

Aby przyspieszyć programowanie, możemy zaimportować wszystkie elementy z biblioteki tkinter za pomocą kodu:

```
from tkinter import *
```

W ten sposób wydłużylibyśmy jednak start naszego programu o zaczytanie wszystkich elementów i metod frameworka. Ze względów wydajnościowych lepiej z niego nie korzystać.

Możemy również wykorzystać kod:

```
import tkinter as tk
```

Nie trzeba wtedy za każdym razem wpisywać „tkinter”, tylko wystarczy „tk” (skrót nazwy frameworku jest ogólnie uznawany za poprawny, więc w razie szukania dodatkowej pomocy w internecie w większości kodów tak będzie wyglądać import frameworku tkinter). Start programu nie będzie trwał zbyt długo nawet na wolniejszym lub starszym sprzęcie.

Tekst do okna aplikacji

Aby dodać tekst do okna aplikacji, możemy stworzyć obiekt typu StringVar z frameworku tkinter. Kod będzie wyglądał następująco:

```
tekst1 = tk.StringVar("Witaj")
```

Etykieta

Następnym krokiem jest utworzenie elementu, który będzie zarządzał obiektem StringVar – Label. Możemy utworzyć taki obiekt na kilka sposobów:

1. Wpisać zwykły tekst na odpowiednim elemencie okna i w odpowiednim elemencie siatki (grida)

   ```
   label1 = tk.Label(okno, textvariable=tekst1)
   label1.grid(row=0,kolumn=0)
   ```

   Grid w tkinterze jest domyślny i dynamiczny – automatycznie dzieli się procentowo na tyle wierszy i kolumn, ile do niego zapiszemy, zaczyna iterowanie kolumn i wierszy od zera.
2. Określić kolor tła i fontu za pomocą konstruktora Label z użyciem angielskich nazw kolorów:

   ```
   label2 = tk.Label(okno, textvariable=tekst1, bg="black",
   fg="white")
   label2.grid(row=1,kolumn=0)
   ```

3. Określić kolor tła i fontu za pomocą konstruktora Label z użyciem kodów heksadecymalnych kolorów (w formacje #RRGGBB):

   ```
   label3 = tk.Label(okno, textvariable=tekst1, bg="#ff0000",
   fg="#00ff00")
   label3.grid(row=2,kolumn=0)
   ```

4. Określić dokładne parametry fontu po przecinku z użyciem parametru font – skorzystanie z domyślnego fontu systemowego:

   ```
   label4 = tk.Label(okno, textvariable=tekst1, font="none 16 bold")
   label4.grid(row=3,kolumn=0)
   ```

5. Określić dokładne parametry fontu po przecinku z użyciem parametru font – z użyciem konkretnego fontu:

   ```
   label5 = tk.Label(okno, textvariable=tekst1, font=("Times New
   Roman", 16 "italic"))
   label5.grid(row=4,kolumn=0)
   ```

```
import tkinter as tk

okno = tk.Tk()
# Dodanie nazwy okna
okno.title("Moja apka")
# dodanie wymiarów okna
okno.geometry("300x400")
# zablokowanie skalowania okna
okno.resizable(False, False)
tekst = tk.StringVar()
tekst.set("Witaj")
# zwykły tekst
label01 = tk.Label(okno, textvariable=tekst)
# ustawienie nowego tekstu na miejscu
label01.grid(row=0,column=0)
# zmiana kolorów tekstu i tła
label02 = tk.Label(okno, textvariable=tekst,
bg="black", fg="white")
label02.grid(row=1,column=0)
```

Rys. 13.5. Kod języka programowania Python – dodanie tekstu w celu komunikacji z użytkownikiem (część 1)

CIEKAWOSTKA

Framework tkinter daje nam pełną kontrolę nad wyświetlanym tekstem. Wielką jego zaletą jest podawanie kolorów w formacie nazw angielskich (np. white, gray, black, blue, red) i w kodzie heksadecymalnym (#ffffff, #aaaaa, #000000, #0000ff, #ff0000). Co więcej, font możemy opisać na kilka sposobów, np. font=”Arial 20 italic” bądź font= („Comic Sans”, 16, „italic”, „bold”). Na rys. 13.5 i 13.6 przedstawiono tylko niewielką część możliwości tkintera.

```
# zmiana kolorów tekstu i tła za pomocą kolorów
zapisanych hexadecymalnie
label03 = tk.Label(okno, textvariable=tekst,
bg="#ff0000", fg="#00ff00")
label03.grid(row=2,column=0)
# zmiana wielkości i pogrubienie tekstu
label04 = tk.Label(okno, textvariable=tekst,
font="none 16 bold")
label04.grid(row=3,column=0)
# dodanie systemowego fontu i ustawienie pochyłej
kursywy tekstu
label05 = tk.Label(okno, textvariable=tekst, font=
("Times New Roman", 16, "italic"))
label05.grid(row=4,column=0)
okno.mainloop()
```

Rys. 13.6. Kod języka programowania Python – dodanie tekstu w celu komunikacji z użytkownikiem (część 2)



Rys. 13.7. Wynik działania programu – dodanie tekstu w celu komunikacji z użytkownikiem

13.4. Przyciski

Tkinter umożliwia w łatwy sposób dodawanie przycisków za pomocą klasy **`Button()`**. Poniższy przykład będzie zgodny z zasadami programowania obiektowego.

Na potrzeby tego zagadnienia utworzymy klasę „Aplikacja". Będzie ona zawierać konstruktor, w którym dodamy i skonfigurujemy wszystkie elementy okna i dwie metody.

1. Konstruktor funkcji będzie pobierał na wejściu obiekt **`tk.Tk()`** – okno naszego programu.
2. Pierwsza metoda (klasy Aplikacja) będzie się nazywać „zmien_tekst" – zmieni zawartość obiektu typu StringVar po jej wywołaniu, nie będzie przyjmować żadnych parametrów (oprócz parametru **`self`** – domyślnego w klasach języka programowania Python).
3. Druga metoda klasy (Aplikacja) będzie przyjmować na wejściu jeden argument – nazwę obiektu typu Label. Zmieni ona tekst wybranego obiektu typu Label z klasy.

Jesteśmy w stanie za pomocą biblioteki tkinter utworzyć przycisk z tekstem i od razu podpiąć jego „kliknięcie" pod odpowiednią funkcję lub metodę. Możemy to zrobić za pomocą argumentu **`command`** na dwa sposoby:

Z użyciem metody lub funkcji bez parametrów:

```
self.b1 = tk.Button(okno,text="Przycisk nr 1",command=self.
zmien_tekst)
self.b1.grid(row=0,column=1)
```

Z użyciem metody lub funkcji z parametrami za pomocą wbudowanej w język Python funkcji **`lambda`**:

```
self.b1 = tk.Button( okno, text = "Przycisk nr 2", command =
lambda: self.zmien_tekst_w(self.label02))
self.b1.grid(row=1,column=1)
```

```python
# klasa używająca frameworka Tkinter
class Aplikacja:
    #pobranie elementu okna w konstruktorze klasy
    def __init__(self, okno):
        self.tekst = tk.StringVar()
        self.tekst.set("Witaj")
        # zwykły tekst
        self.label01 = tk.Label(okno,
        textvariable=self.tekst)
        # ustawienie nowego tekstu na miejscu
        self.label01.grid(row=0,column=0)
```

Rys. 13.8. Kod języka programowania Python – tworzenie przycisków (część 1)

```
        self.b1 = tk.Button(okno,text="Przycisk nr
        1",command=self.zmien_tekst)
        self.b1.grid(row=0,column=1)

        # drugi tekst
        self.label02 = tk.Label(okno,text="Zmień
        mnie")
        # ustawienie nowego tekstu na miejscu
        self.label02.grid(row=1,column=0)

        self.b1 = tk.Button(okno,text="Przycisk nr
        2",command=lambda: self.zmien_tekst_w(self.
        label02))
        self.b1.grid(row=1,column=1)
```

Rys. 13.8. Cd. Kod języka programowania Python – tworzenie przycisków (część 1)

```
    def zmien_tekst(self):
        self.tekst.set("Tekst został zmieniony")

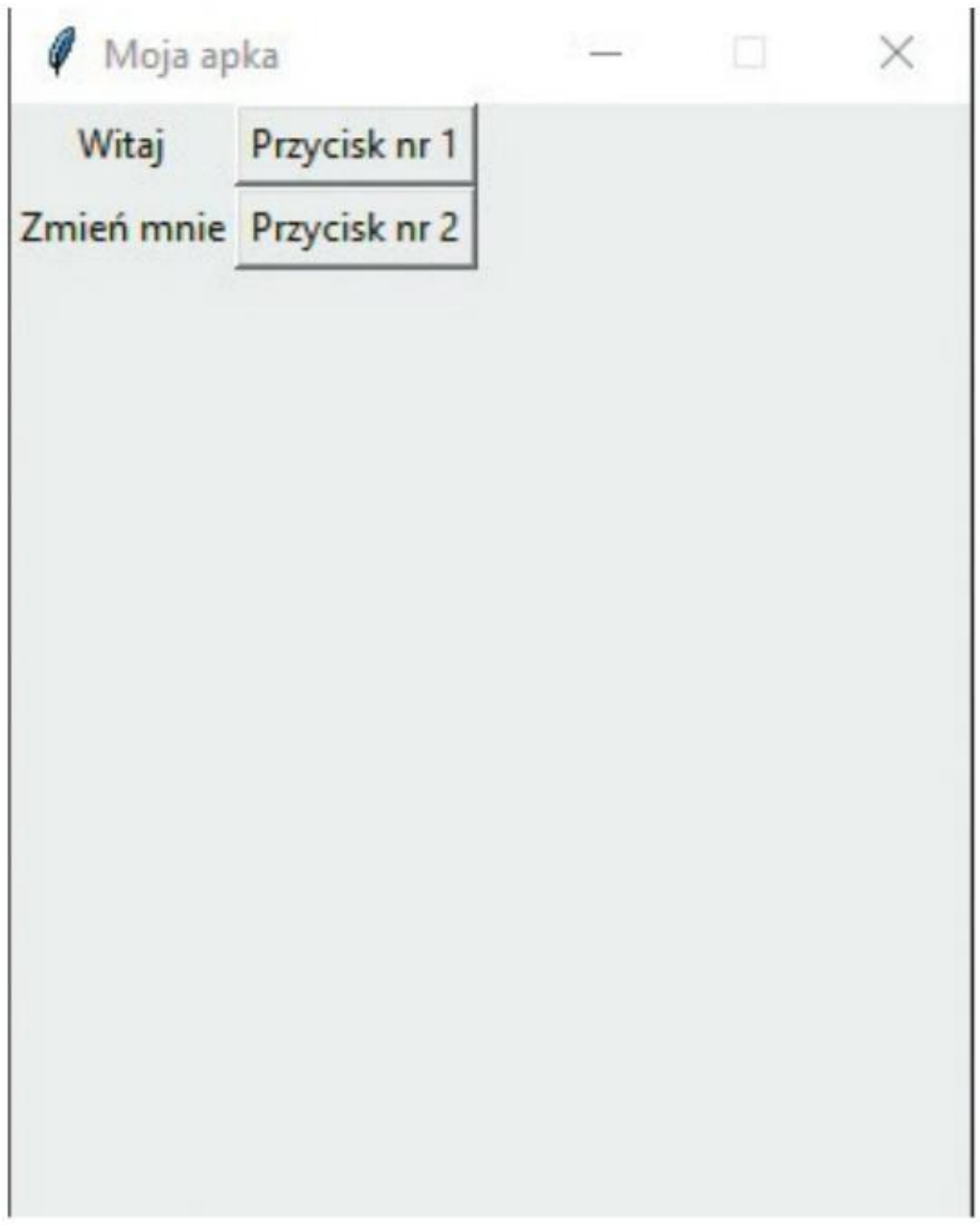
    def zmien_tekst_w(self, label):
        label.config(text="Tekst został zmieniony
        2")

okno = tk.Tk()
# Dodanie nazwy okna
okno.title("Moja apka")
# dodanie wymiarów okna
okno.geometry("300x400")
# zablokowanie skalowania okna
okno.resizable(False, False)

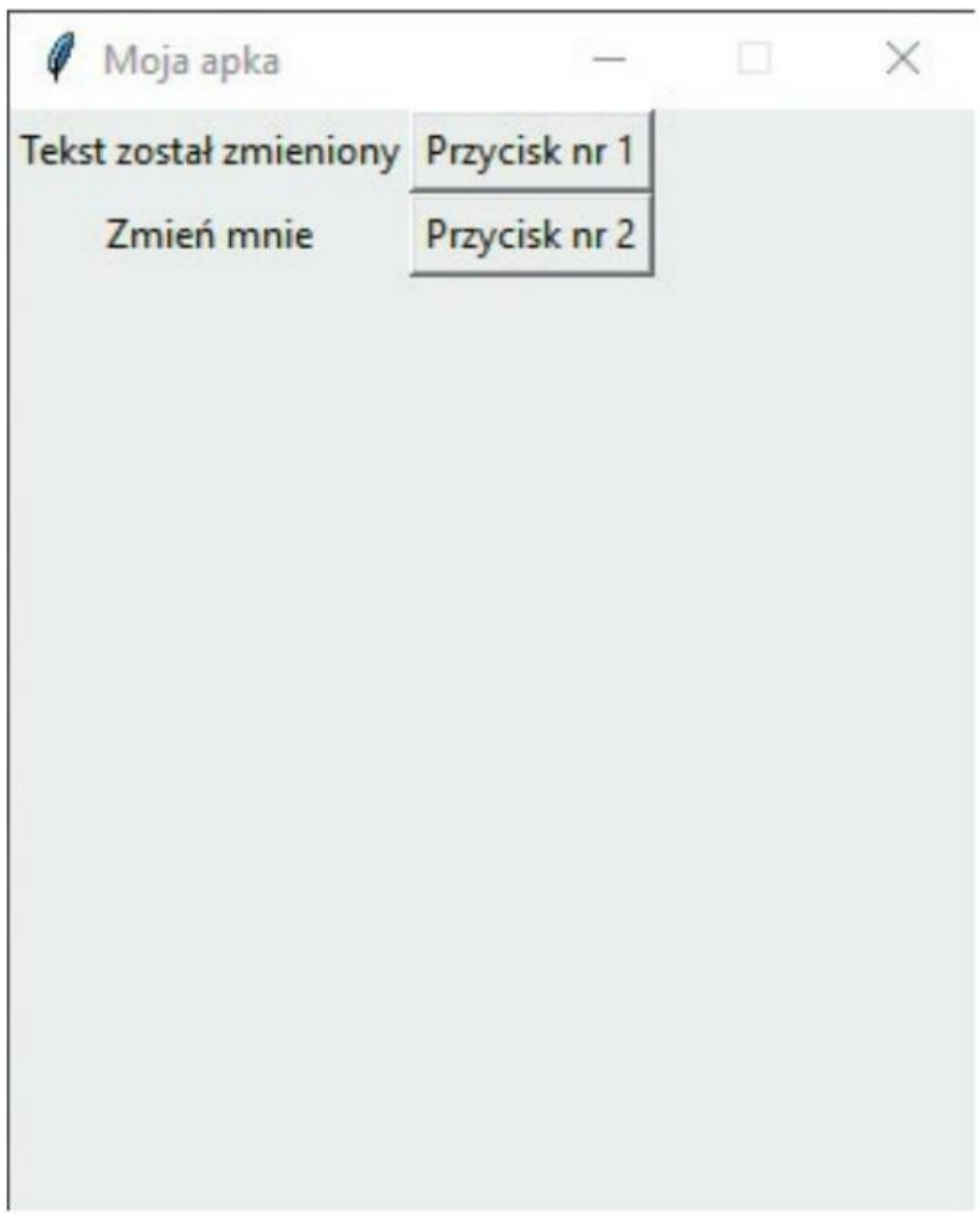
app = Aplikacja(okno)

okno.mainloop()
```

Rys. 13.9. Kod języka programowania Python – tworzenie przycisków (część 2)



Rys. 13.10. Wynik działania programu – przed kliknięciem przycisków



Rys. 13.11. Wynik działania programu po kliknięciu przycisku nr 1



Rys. 13.12. Wynik działania programu po kliknięciu przycisku nr 2

13.5. Wejście użytkownika

Aby wprowadzić wejście od użytkownika, posłużymy się prostym obiektem o nazwie „Entry", zawartym w naszym frameworku.

Obiekt typu **`Entry`** ma wymagany parametr – nazwę okna, w którym ma się znajdować. Ma również opcjonalne parametry, takie jak:

- **textvariable** – nazwa zmiennej StringVar, w której będzie przechowywana zawartość elementu Entry;
- **font** – opis fontu, rozmiaru i kursywy znaków, które będziemy wpisywali do pola Entry, np. **`font = ('none', 9, 'italic')`** (będzie to czcionka domyślna systemu o rozmiarze 9 i pochylona w prawo);
- **show** – sposób wyświetlania znaków w polu, np. dla **`show="*"`** wpisany tekst „abcde" będzie widoczny w polu elementu Entry jako „*****".

```
import tkinter as tk

# klasa używająca frameworka Tkinter
class Aplikacja:
    #pobranie elementu okna w konstruktorze klasy
    def __init__(self, okno):
        #utworzenie zmiennych tekstowych
        self.tekst1 = tk.StringVar()
        self.tekst2 = tk.StringVar()
        # zwykły tekst
        self.label01 = tk.Label(okno, text="Podaj
        liczbę")
        self.label01.grid(row=0, column=0)
        # wejście od użytkownika
        self.entry01 = tk.Entry(okno,
        textvariable=self.tekst1, font=('none',9,
        'italic'))
        # ustawienie nowego entry na miejscu
        self.entry01.grid(row=0,column=1)

        #Dodanie przycisku z wywołaniem metody
        self.b1 = tk.Button(okno,text="Przekaż
        liczbę",command=lambda: self.zmien_tekst
        (self.label03, self.tekst1, "Podana liczba:
        "))
        self.b1.grid(row=0,column=2)
```

Rys. 13.13. Kod języka programowania Python – tworzenie wejścia dla użytkownika (część 1)

```
# zwykły tekst
self.label02 = tk.Label(okno, text="Podaj
hasło")
self.label02.grid(row=1, column=0)

# drugie wejście tekstowe, tym razem z
ukrytym wpisywaniem
self.entry02 = tk.Entry(okno,
textvariable=self.tekst2, show="*")
# ustawienie nowego tekstu na miejscu
self.entry02.grid(row=1,column=1)

#Dodanie przycisku z wywołaniem metody
self.b1 = tk.Button(okno,text="Przekaż
hasło",command=lambda: self.zmien_tekst
(self.label04, self.tekst2, "Hasło to: "))
self.b1.grid(row=1,column=2)

# trzeci tekst
self.label03 = tk.Label(okno,text="Zmień
mnie")
# ustawienie nowego tekstu na miejscu
self.label03.grid(row=2,column=1)
```

Rys. 13.14. Kod języka programowania Python – tworzenie wejścia dla użytkownika (część 2)

```
        # czwarty tekst
        self.label04 = tk.Label(okno,text="Hasło
        to")
        # ustawienie nowego tekstu na miejscu
        self.label04.grid(row=3,column=1)

    def zmien_tekst(self, label, textvar, text):
        #self.tekst.set("Tekst został zmieniony")
        label.config(text=(str(text)+ str(textvar.
        get())))


okno = tk.Tk()
# Dodanie nazwy okna
okno.title("Moja apka")
# dodanie wymiarów okna
okno.geometry("300x400")
# zablokowanie skalowania okna
okno.resizable(False, False)

app = Aplikacja(okno)

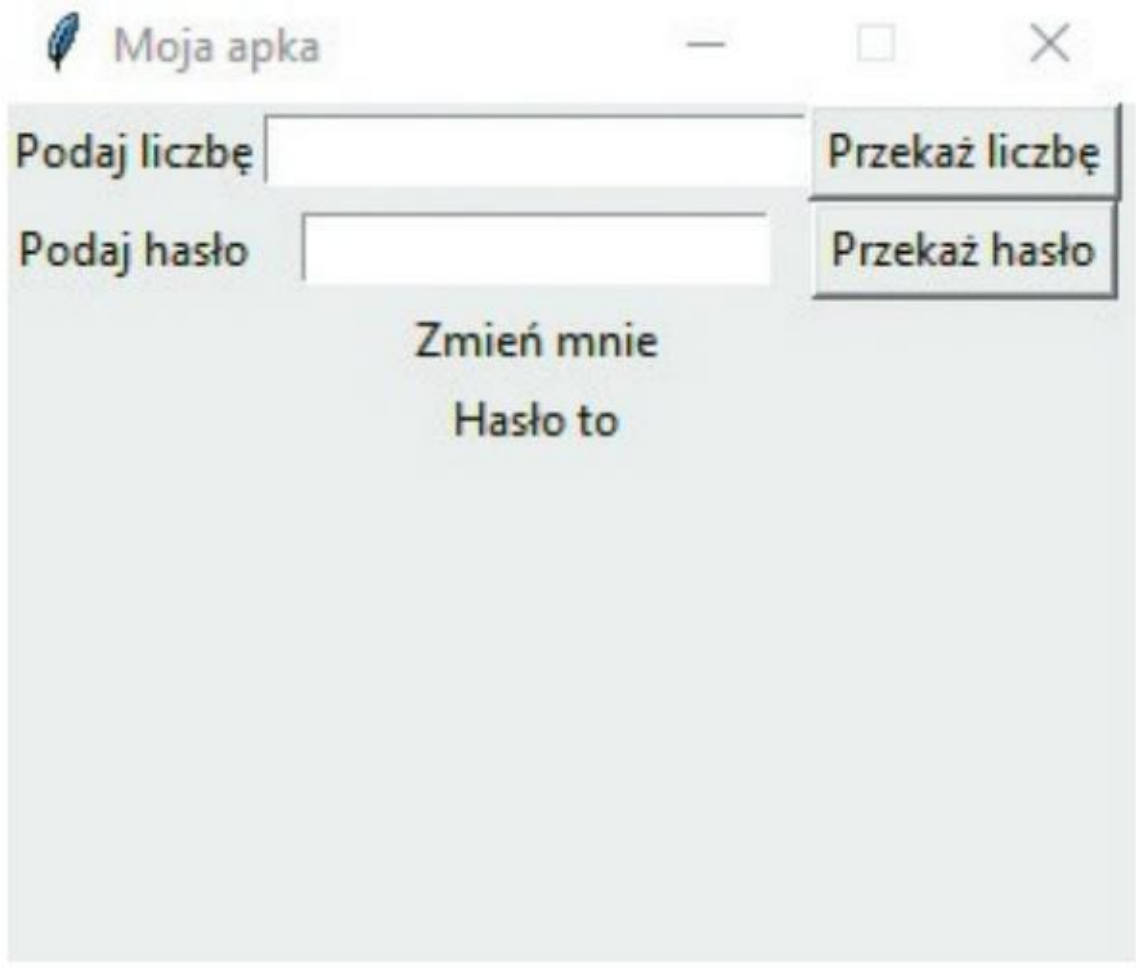
okno.mainloop()
```

Rys. 13.15. Kod języka programowania Python – tworzenie wejścia dla użytkownika (część 3)

CIEKAWOSTKA

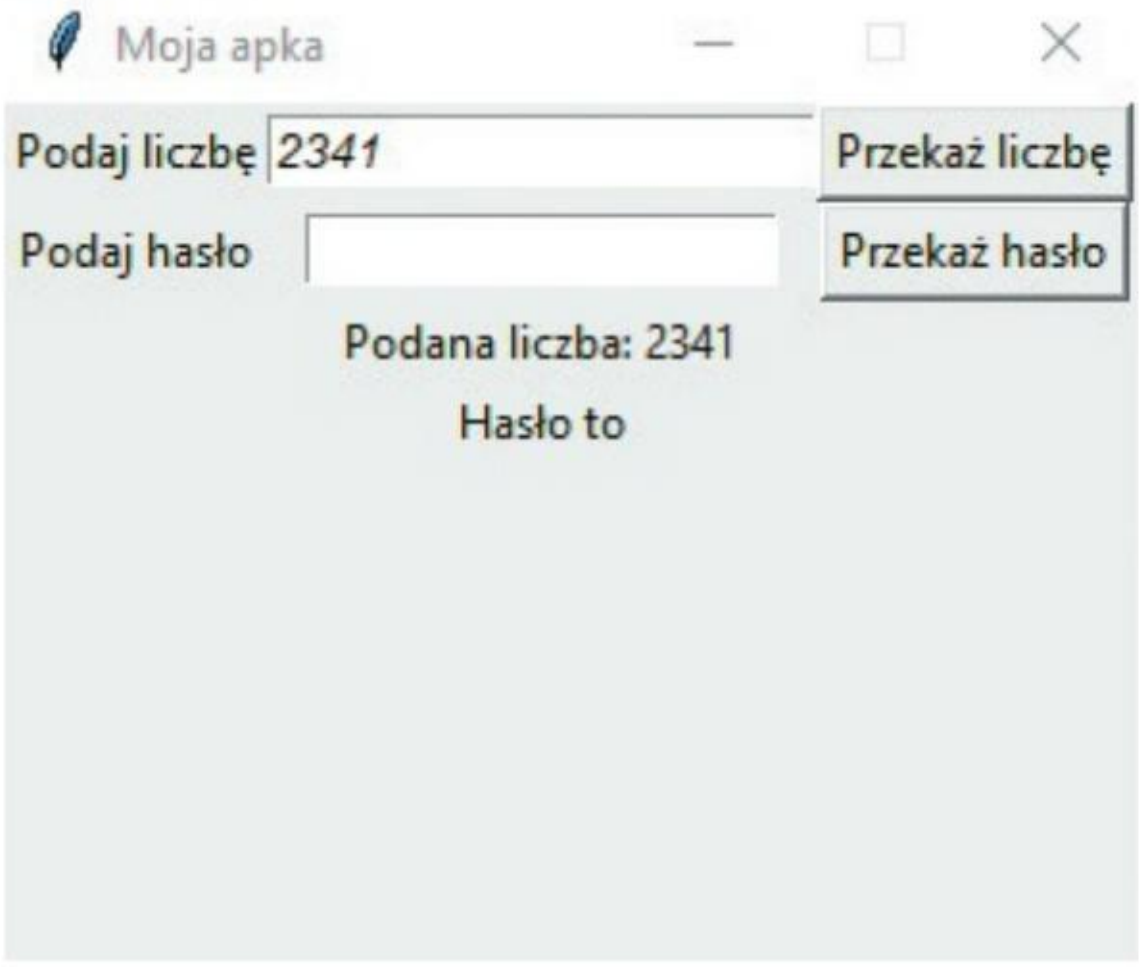
W obiektach frameworka tkinter typu Entry i Button można sterować tekstem w taki sam sposób, jak w obiektach typu Label, czyli z użyciem argumentów konstruktora font, bg, fg. Przykładowo, aby utworzyć kolorowy przycisk, wystarczy napisać:

```
tk.Button(okno, text="Czerwony tekst", fg="#FF0000",
bg="#000000", font=(„Comic Sans", 16, „italic")
```



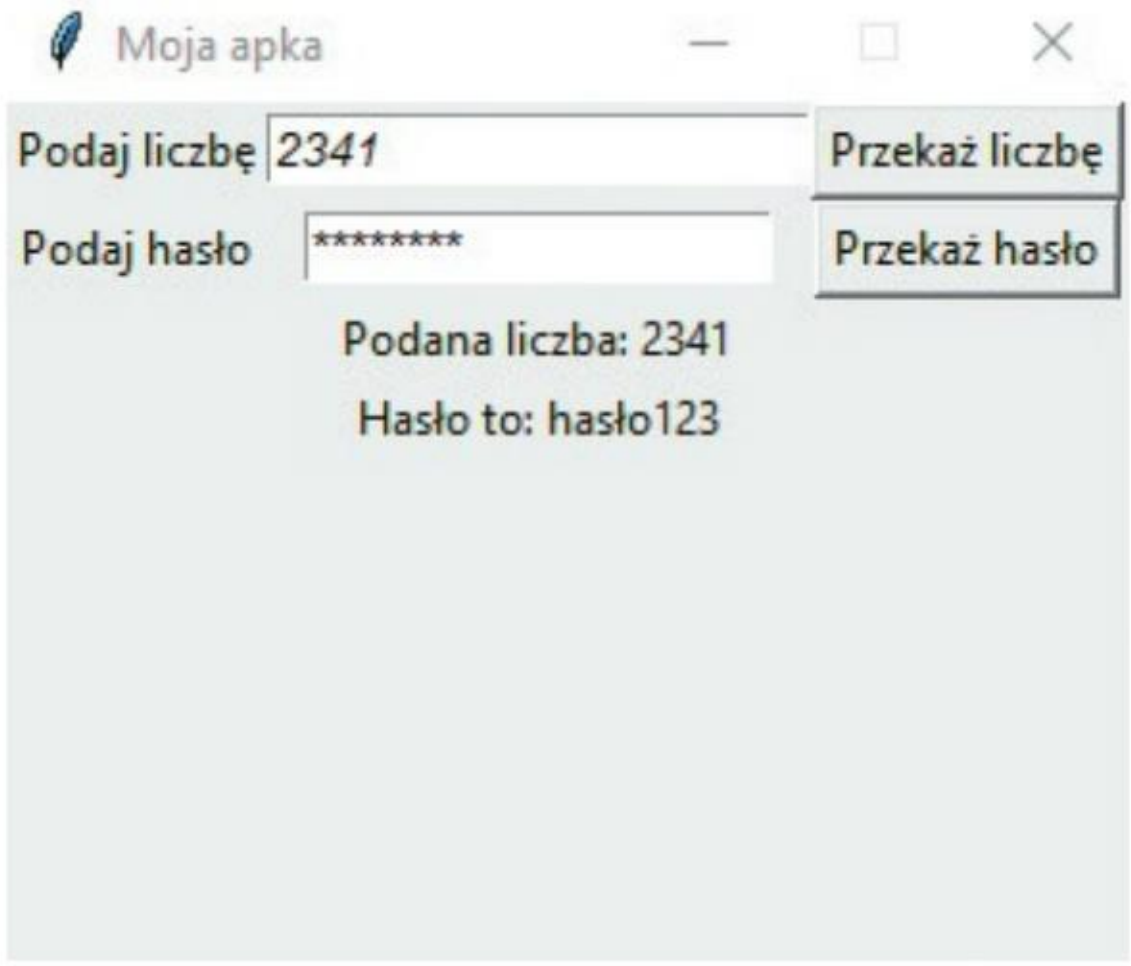
Rys. 13.16. Wynik działania programu przed kliknięciem przycisków

Na rys. 13.16 przedstawiono okno programu. Z lewej strony widać na nim dwie kolumny obiektów tekstowych typu Label. W środkowej części znajdują się dwa obiekty typu Entry, a pod nimi dwa kolejne obiekty typu Label. W lewej części są dwa przyciski, za pomocą których możemy uaktualniać zawartość dwóch najniższych obiektów Label z drugiej kolumny.



Rys. 13.17. Wynik działania programu po kliknięciu przycisku „Przekaż liczbę”

Na rys. 13.17 widać, że w pierwszym obiekcie typu Entry tekst jest trochę większy niż reszta tekstu w aplikacji i zapisany kursywą. Po naciśnięciu przycisku „Przekaż liczbę” zawartość Labela z tekstem „Zmień mnie” została zaniesiona na „Podana liczba: 2341”.



Rys. 13.18. Wynik działania programu po kliknięciu przycisku „Przekaż hasło”

Po wpisaniu tekstu w drugie pole typu Entry podawane znaki są wyświetlane jako „*”. Po wpisaniu w to pole tekstu „hasło123” i kliknięciu przycisku „Przekaż hasło” tekst w najniżej położonym obiekcie typu Label zamieni się z „Hasło to” na „Hasło to: hasło123”.

13.6. Zaawansowane wejście użytkownika, obsługa klawiatury i zdarzenia myszy

Pomimo szybkiego i prostego sposobu tworzenia gotowych elementów we frameworku tkinter czasem brakuje pewnych funkcji – np. obiekt typu Entry różni się od webowego „inputa” tym, że nie możemy wpisać do niego tekstu zastępczego podpowiadającego użytkownikowi, co ma wpisać. Z pomocą przychodzą obiektowość i dziedziczenie. Możemy utworzyć nową klasę, dziedziczącą po klasie Entry i mającą w swoim wnętrzu dodatkowy tekst, podpowiadający użytkownikowi, co powinno znaleźć się w środku.

Aby pole podpowiadające użytkownikowi, co powinien wpisać, mogło funkcjonować, trzeba użyć „bindowania”, czyli przejęcia informacji z myszki / klawiatury. Framework tkinter daje nam wiele możliwości obsługi klawiatury i myszy. Najprostszą metodą jest wywołanie w konstruktorze klasy poniższej formuły:

```
self.bind("<nazwa_zdarzenia>", self.metoda)
```

lub

```
self.bind("<nazwa_zdarzenia", funkcja)
```

Taka instrukcja wywoła zapisaną jej funkcję / metodę, gdy dojdzie do zdarzenia. Przykładowo:

- **<Button>** – wciśnięcie przycisku myszy;
- **<Motion>** – gdy w trakcie przyciśnięcia myszy użytkownik nią poruszy;

- **<Enter>** – gdy mysz najedzie na dany element;
- **<Key>** – gdy jakikolwiek klawisz klawiatury zostanie wciśnięty;
- **<Space>** – gdy zostanie wciśnięta spacja;
- **"a"** – wciśnięcie litery „a" wywoła funkcję / metodę.

```
import tkinter as tk

# prosty słownik przechowujący dane uwierzytelniania
a = {"admin": "123", "user": "haslo"}


# tworzymy własną klasę dziedziczącą po klasie Entry
class EntryRozszerzone(tk.Entry):
    # tworzymy konstruktor naszej klasy
    def __init__(self, master=None,
    tekst_zastepczy="podaj liczbe", color='grey'):
        super().__init__(master, width=26)

        # tworzymy zmienną wyświetlającą się, gdy
        input jest pusty
        self.tekst_zastepczy = tekst_zastepczy
        self.tekst_zastepczy_color = color
        self.default_fg_color = self['fg']

        # sprawdzamy, czy użykownik wcisnął
        jakikolwiek przycisk - wtedy nasz tekst
        zniknie
        self.bind("<Key>", self.foc_in)

        # sprawdzamy, czy użytkownik opuścił pole
        self.bind("<Leave>", self.foc_out)
```

Rys. 13.19. Kod języka programowania Python – tworzenie zaawansowanego wejścia dla użytkownika (część 1)

```
            self.put_tekst_zastepczy()

        def put_tekst_zastepczy(self):
            self.delete('0', 'end')
            self.insert(0, self.tekst_zastepczy)
            self['fg'] = self.tekst_zastepczy_color

        def foc_in(self, *args):
            if self['fg'] == self.tekst_zastepczy_color:
                self.delete('0', 'end')
                self['fg'] = self.default_fg_color

        def foc_out(self, *args):
            if not self.get():
                self.put_tekst_zastepczy()

    # klasa używająca frameworka Tkinter
    class Aplikacja:
        #pobranie elementu okna w konstruktorze klasy
        def __init__(self, okno):
            #utworzenie zmiennych tekstowych
            self.tekst1 = tk.StringVar()
            self.tekst2 = tk.StringVar()
            # dodanie pustych elementów, aby utworzyć
            odstępy pomiędzy elementami na gridzie
            self.label01 = tk.Label(okno,
            text="                              ")
```

Rys. 13.20. Kod języka programowania Python – tworzenie zaawansowanego wejścia dla użytkownika (część 2)

```
        self.label01.grid(row=0, column=0)
        self.label01 = tk.Label(okno,
        text="                    ")
        self.label01.grid(row=2,column=0)
        self.label01 = tk.Label(okno,
        text="                    ")
        self.label01.grid(row=4, column=0)
        self.label01 = tk.Label(okno,
        text="                    ")
        self.label01.grid(row=6, column=0)

        # wejście od użytkownika na nazwę
        self.entry01 = EntryRozszerzone(okno,
        tekst_zastepczy="Podaj nazwę użytkownika",)

        # ustawienie nowego entry na miejscu
        self.entry01.grid(row=1,column=1)

        # wejście na hasło od użytkownika
        self.entry02 = EntryRozszerzone(okno,
        tekst_zastepczy="Podaj hasło użytkownika")
        # ustawienie nowego entry na miejscu
        self.entry02.grid(row=3,column=1)
```

Rys. 13.21. Kod języka programowania Python – tworzenie zaawansowanego wejścia dla użytkownika (część 3)

```
        # Dodanie przycisku z wywołaniem metody
        self.b1 = tk.Button(okno,text="Zaloguj się",
        command=lambda: self.zmien_tekst(self.
        label01, self.entry01.get(), self.entry02.
        get()))
        self.b1.grid(row=5,column=1)

        # pierwszy tekst
        self.label01 = tk.Label(okno,text="Czekam
        za zalogowanie")
        # ustawienie nowego tekstu na miejscu
        self.label01.grid(row=7,column=1)


    def zmien_tekst(self, label, user, passwd):
        if user in a:
            pass
            if passwd == a[user]:
                label.config(text="Użytkownik
                zalogowany")
            else:
                label.config(text="Złe hasło")
        else:
            label.config(text="Użytkownik nie
            istnieje")
```

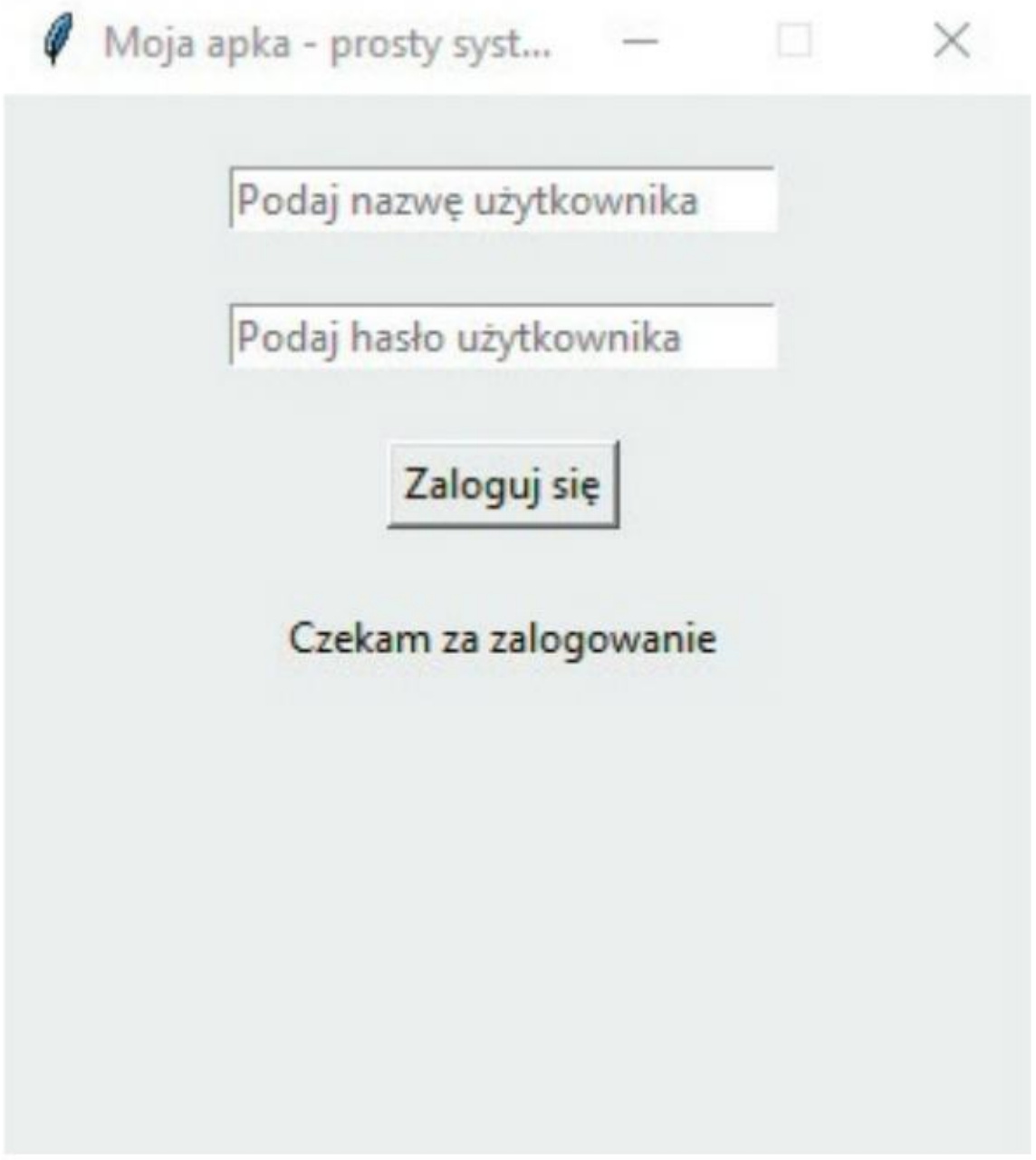
Rys. 13.22. Kod języka programowania Python – tworzenie zaawansowanego wejścia dla użytkownika (część 4)

```
okno = tk.Tk()
# Dodanie nazwy okna
okno.title("Moja apka - prosty system logowania")
# dodanie wymiarów okna
okno.geometry("300x400")
# zablokowanie skalowania okna
okno.resizable(False, False)

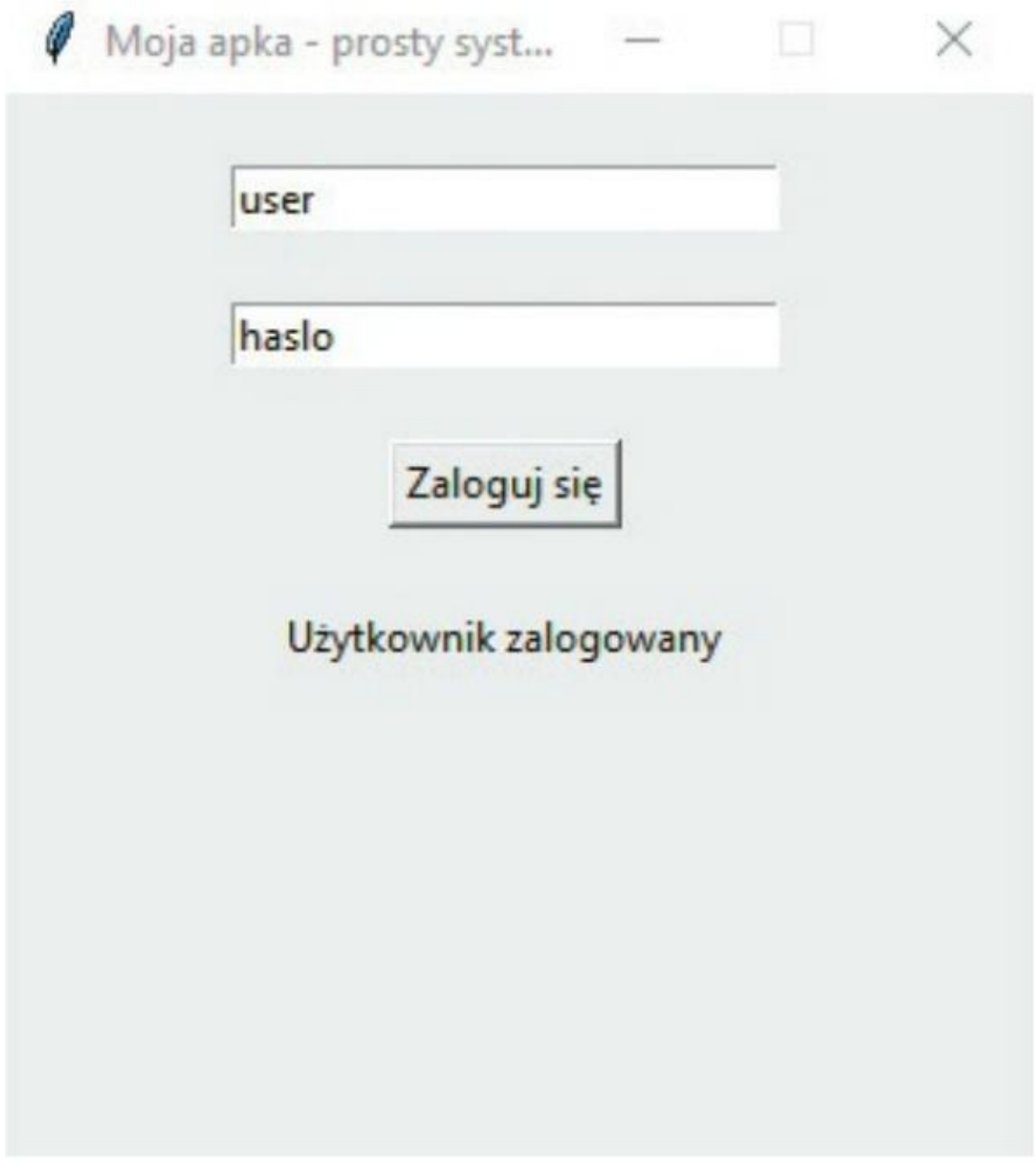
app = Aplikacja(okno)

# start programu
okno.mainloop()
```

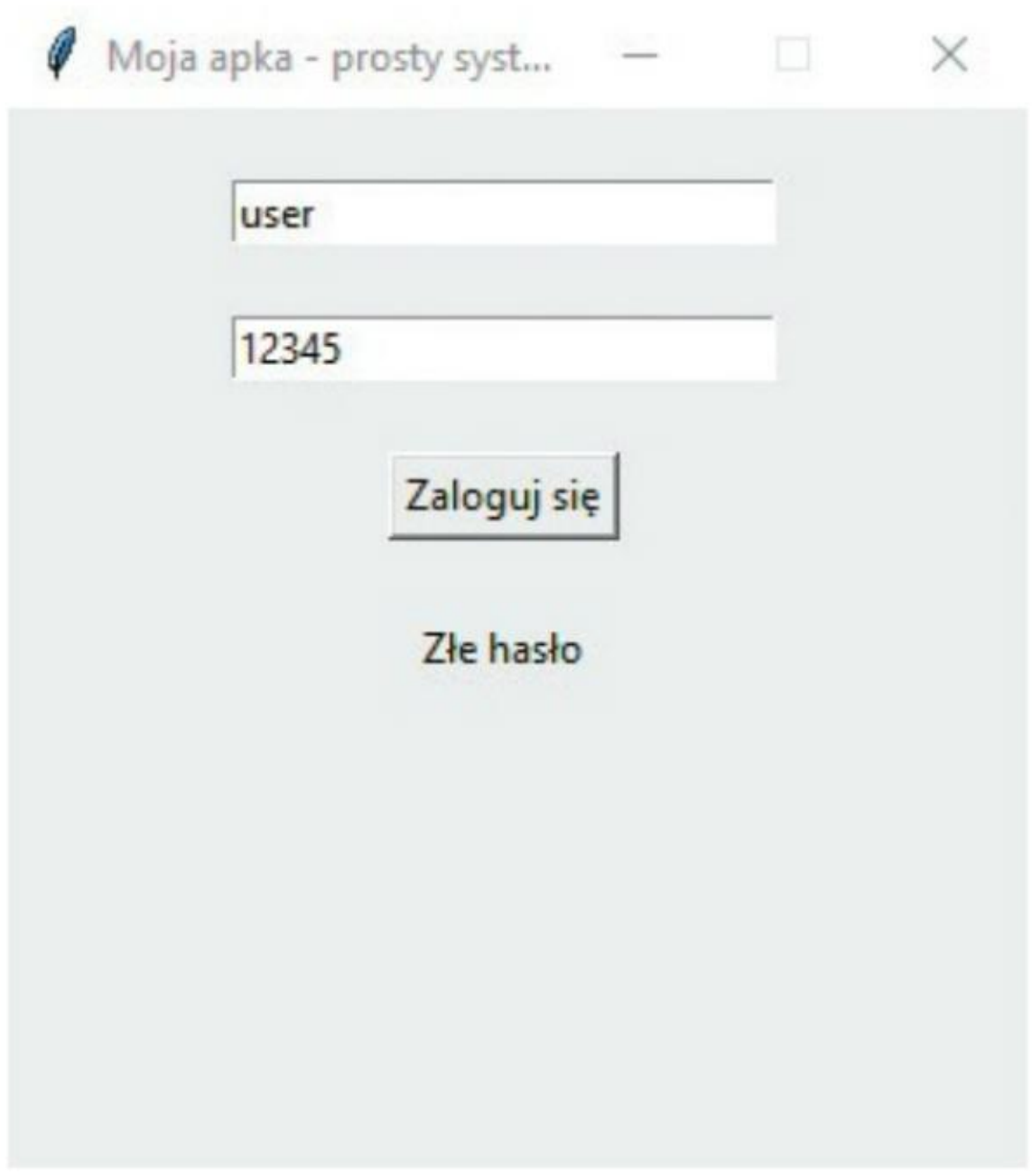
Rys. 13.23. Kod języka programowania Python – tworzenie zaawansowanego wejścia dla użytkownika (część 5)



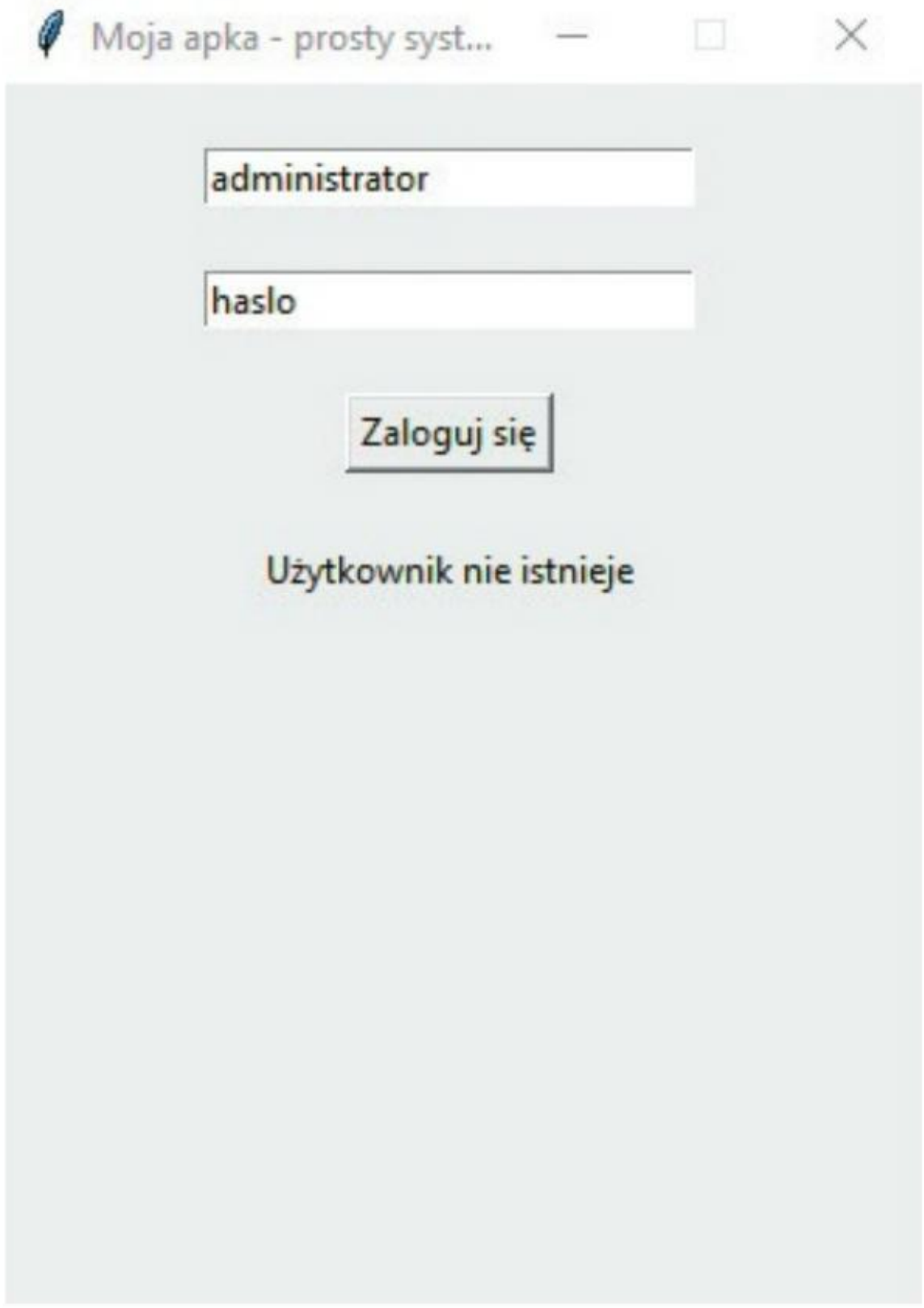
Rys. 13.24. Wynik działania programu przed kliknięciem przycisków



Rys. 13.25. Wynik działania programu po wpisaniu błędnego hasła



Rys. 13.26. Wynik działania programu po wpisaniu błędnej nazwy użytkownika



Rys. 13.27. Wynik działania programu po wpisaniu prawidłowych danych przez użytkownika

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień trzy sposoby utworzenia elementu Label.
2. Opisz proces tworzenia okna w frameworku tkinter.
3. Co możemy zrobić, aby zabronić użytkownikowi skalowania okna naszej aplikacji?
4. W jaki sposób można utworzyć „przycisk” we frameworku tkinter?
5. W jaki sposób możemy pozyskać tekst od użytkownika?
6. Jaki typ obiektu przechowuje pole typu Entry?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz klasę RozszerzonyButton na bazie klasy Button z frameworku tkinter, który zmienia kolor tła przycisku na czerwony w momencie najechania kursora myszy na obiekt.
2. Napisz z użyciem obiektów Button grę w kółko i krzyżyk.
   Podpowiedź: element Button może mieć tekst wpisany jako element „StringVar" przypisany do argumentu konstruktora „textvariable" (tak jak w rozdziale 13.4 z obiektami typu Label).
3. Stwórz z użyciem przycisków Button i obiektu typu Entry prosty kalkulator. Powinien on zawierać możliwość kliknięcia cyfr od 0 do 9 i zawierać operacje matematyczne, takie jak:
   - dodawanie,
   - odejmowanie,
   - mnożenie,
   - dzielenie,
   - reszta z mnożenia,
   - pierwiastek (math.sqrt(x)),
   - potęgę (x**y),
   - wartość absolutną (math.abs(x)).

14 Programowanie desktopowych aplikacji okienkowych za pomocą języka programowania Python

ZAGADNIENIA

- Planowanie aplikacji
- Pisanie aplikacji na bazie poznanych wcześniej elementów interfejsu użytkownika
- Eksport aplikacji do pliku .exe

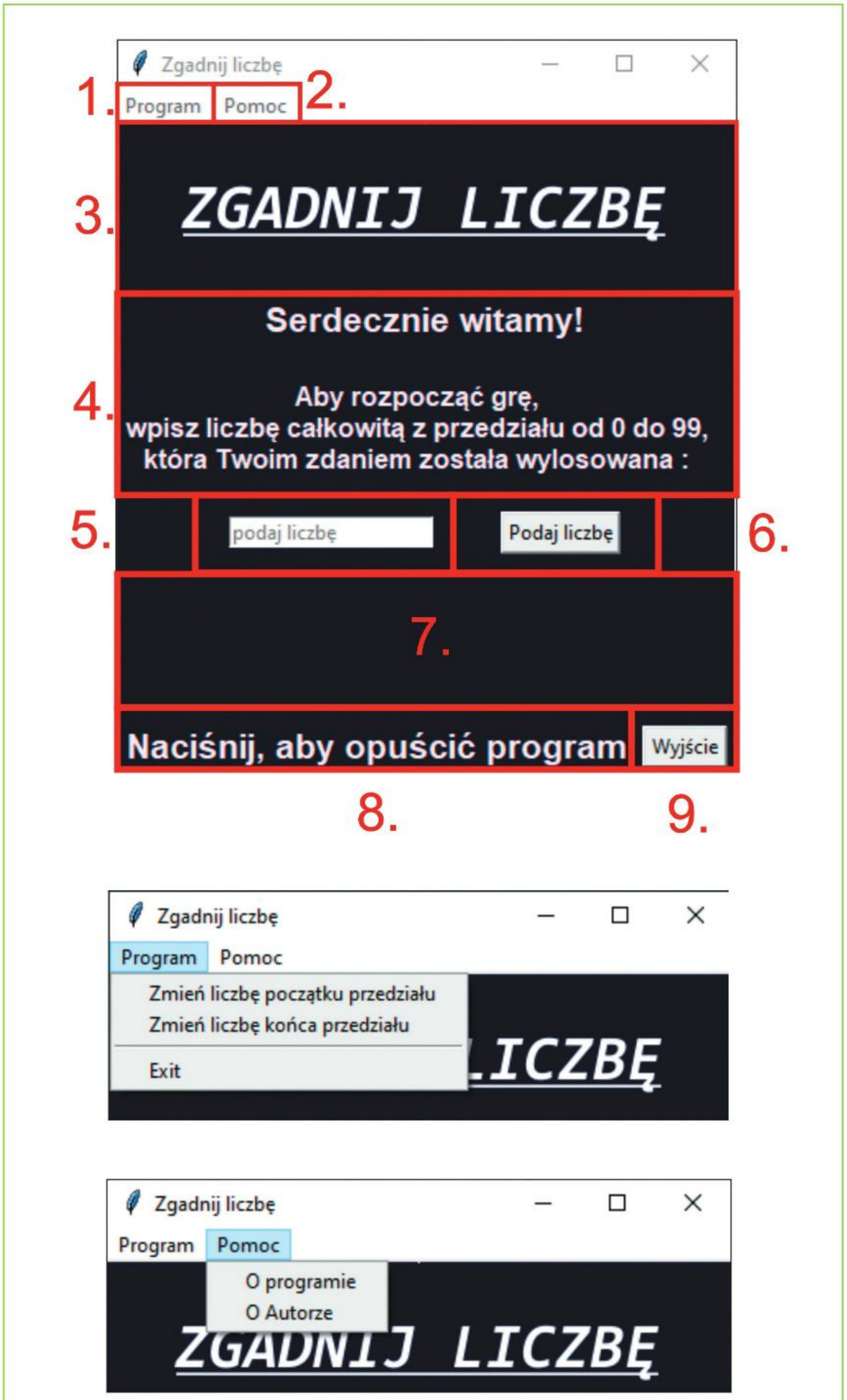
14.1. Planowanie aplikacji

Utworzoną w pierwszym rozdziale aplikację konsolową Gra edukacyjna Zgadnij Liczbę nieco zmienimy i utworzymy do niej interfejs użytkownika za pomocą frameworku tkinter z użyciem obiektowego podejścia języka programowania Python.

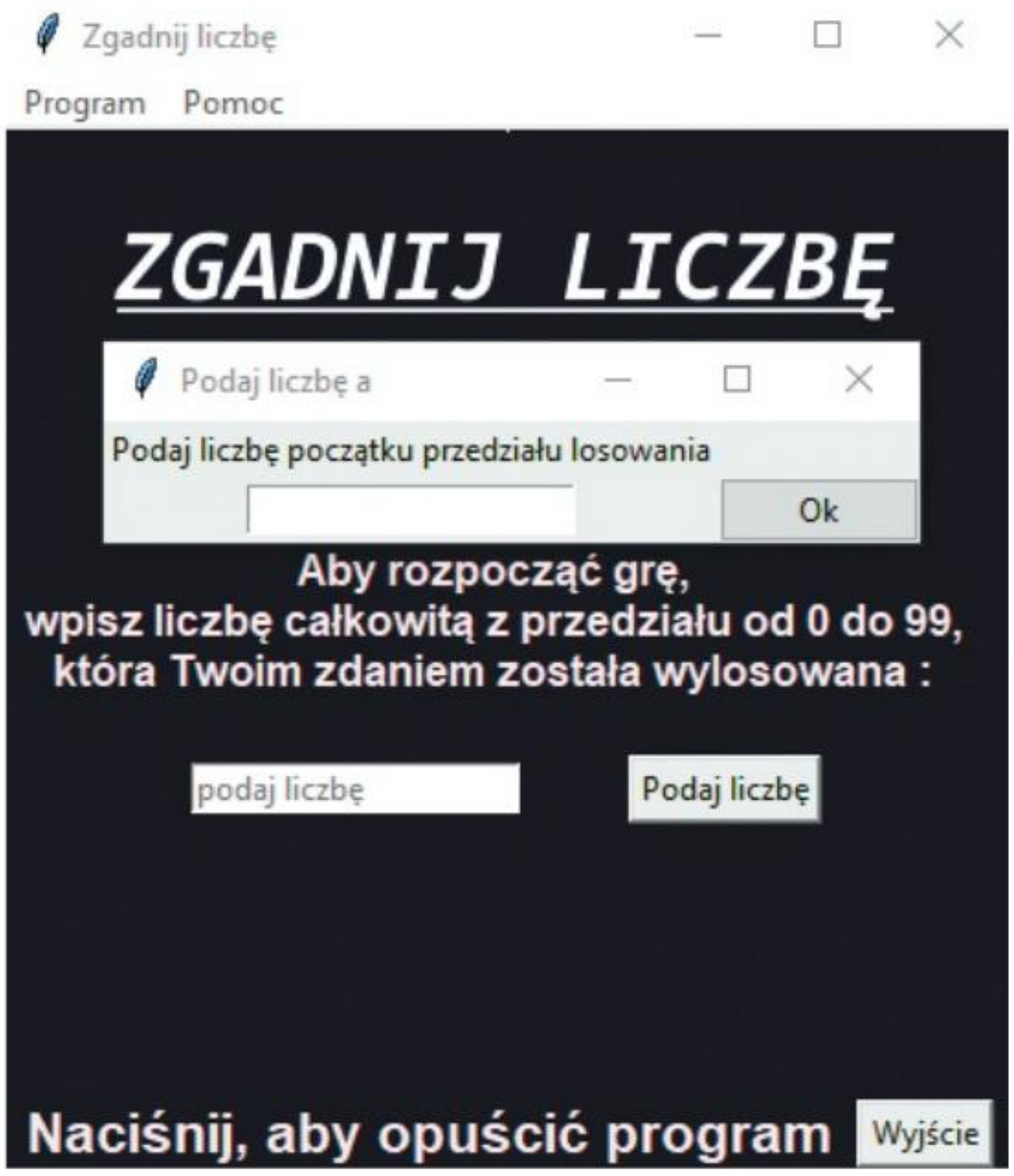
Aplikacja będzie wyglądała następująco (zob. rys. 14.1):

1. Przycisk „Program" – na pasku menu, który po rozwinięciu i użyciu będzie tworzyć okna modalne.
2. Przycisk „Pomoc" – na pasku menu, który po rozwinięciu i użyciu będzie tworzyć okna modalne.
3. Baner – zdjęcie w tle.
4. Opis gry – wyświetlony za pomocą obiektów Label.
5. Wejście dla użytkownika, rozszerzone Entry (napisanie przez nas w poprzednim rozdziale).
6. Przycisk „Podaj liczbę" – wywołujący metody gry i sprawdzający poprawność liczby wpisanej przez użytkownika.
7. Miejsce, w którym będzie przebiegała interakcja z użytkownikiem – wyświetlane informacje, czy wpisana liczba jest większa, mniejsza, czy równa szukanej liczbie – lub powiadomienie, że znaki wprowadzone przez użytkownika nie są liczbą.
8. Informacja o dodatkowym sposobie opuszczenia gry.
9. Przycisk kończący pracę programu.

Po kliknięciu w **`Program ->`** „Zmień liczbę początku przedziału" z rozwijanego menu powinno się pojawić okno modalne – czyli takie, które zabroni użytkownikowi interakcji z wcześniej otwartymi oknami naszej aplikacji tak długo, aż będzie ono otwarte. Będzie ono wyglądało tak, jak to przedstawiono na rys. 14.2.

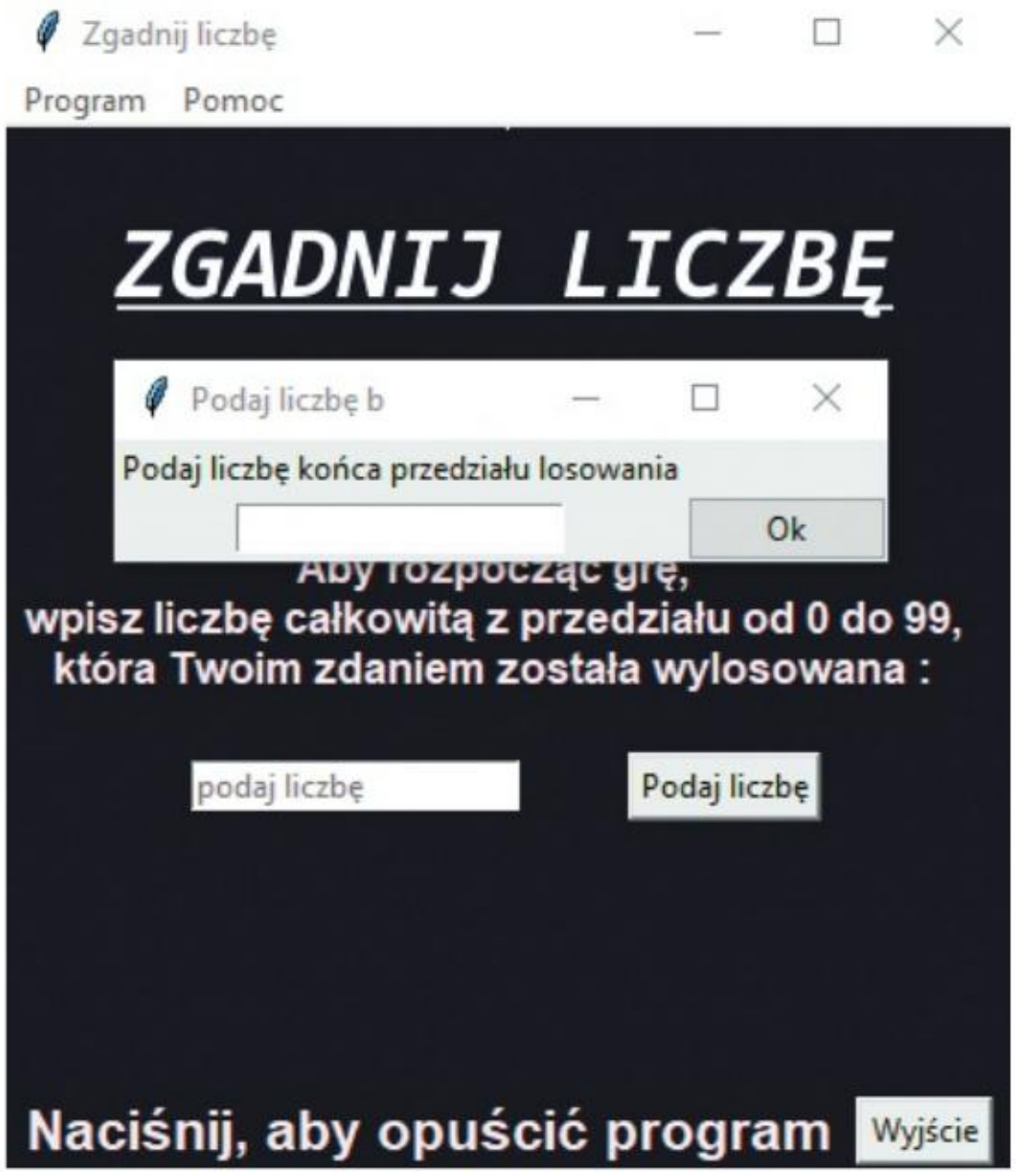


Rys. 14.1. Ostateczny wygląd aplikacji



Rys. 14.2. Okno modalne – Podaj liczbę początku przedziału losowania

Po kliknięciu opcji „Zmień liczbę końca przedziału" z zakładki „Program" (w rozwijanym menu) powinno się wyświetlić okno modalne, jak na rys. 14.3.



Rys. 14.3. Okno modalne – Podaj liczbę końca przedziału losowania

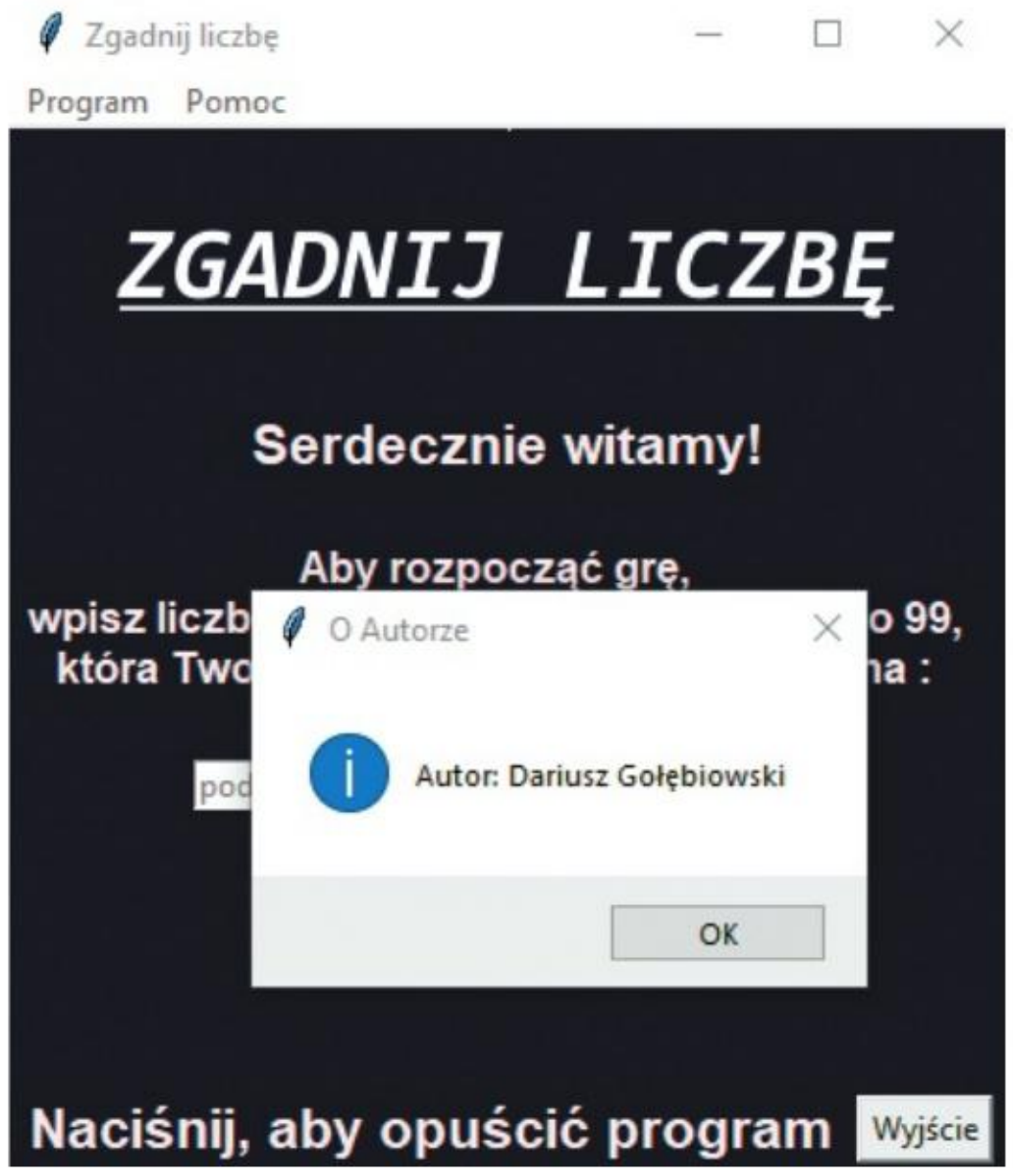
Po wybraniu opcji „Exit" z zakładki „Program" z menu aplikacja zostanie zamknięta.

Po kliknięciu w **`Pomoc ->`** „O programie" (w rozwijanym menu) powinno się pojawić proste, domyślne, systemowe okno (rys. 14.4).



Rys. 14.4. Okno modalne – informacje o programie

W rozwijanym menu, po wybraniu opcji w **`Pomoc ->`** O autorze, pojawi się okno informujące o autorze aplikacji (rys. 14.5).



Rys. 14.5. Okno modalne – informacje o autorze

14.2. Pisanie aplikacji na bazie poznanych wcześniej elementów interfejsu użytkownika

Aplikację zaczniemy pisać od zaimportowania kilku bibliotek i utworzenia niezbędnych zmiennych globalnych, które ułatwią pracę.

Musimy zaimportować biblioteki:

- tkinter – framework do tworzenia aplikacji okienkowych;
- random – dzięki niej możemy losować liczbę do zgadnięcia;
- PIL – do przetworzenia zdjęcia, które wyświetlimy w aplikacji.

Aby zainstalować na komputerze bibliotekę PIL, musisz wpisać w konsoli następujące polecenie:

```
pip install Pillow
```

Potrzebujemy również globalnej funkcji, którą będziemy mogli wywołać w każdym miejscu programu, aby ułatwić losowanie szukanej liczby.

```
import tkinter as tk
from tkinter import ttk
from tkinter.messagebox import showinfo
import random

from PIL import Image, ImageTk

aLiczbaJestWieksza = "\nSzukana liczba jest
większa,\nspróbuj ponownie wpisać jakąś liczbę\n"
bLiczbaJestMniejsza = "\nSzukana liczba jest
mniejsza,\nspróbuj ponownie wpisać jakąś liczbę\n"
cZgadzaSie = "\nBrawo, gratuluję, zgadłeś
liczbę!\nTo faktycznie jest liczba: "
```

Rys. 14.6. Aplikacja – import potrzebnych bibliotek, utworzenie zmiennych globalnych dla aplikacji i stworzenie funkcji losującej szukaną liczbę

```
a = 0
b = 99
n = random.randint(a, b)

text_color = "#ffeeee"
bg_color = "#111133"


def global_n_randint(a1, b1):
    global n
    n = random.randint(a1, b1)
```

Rys. 14.6. Cd. Aplikacja – import potrzebnych bibliotek, utworzenie zmiennych globalnych dla aplikacji i stworzenie funkcji losującej szukaną liczbę

Następnym krokiem będzie dopisanie klasy, która obsłuży okno modalne „Podaj liczbę początku przedziału losowania". Musimy utworzyć tam jeden obiekt typu Label, jeden obiekt typu Entry i element Button, wywołujący metodę „sprawdzenie_poprawnosci".

```
class podaj_liczbe_poczatek:
    def __init__(self):
        self.okno = tk.Toplevel()
        self.okno.wm_title("Podaj liczbe a")
        self.okno.grab_set()
        self.l = tk.Label(self.okno, text="Podaj
        liczbę początku przedziału losowania")
        self.l.grid(row=0, column=0)

        self.text1 = tk.Entry(self.okno)
        self.text1.grid(row=1,column=0)
        self.b = ttk.Button(self.okno, text="Ok",
        command=self.sprawdzanie_poprawnosci)
        self.b.grid(row=1, column=1)
        # okno.grab_release()
```

Rys. 14.7. Utworzenie klasy okna modalnego, pobierającej od użytkownika liczbę i sprawdzającej jej poprawność – konstruktor

Metoda „sprawdzenie_poprawnosci" musi się upewnić, że użytkownik wpisał liczbę, i sprawdzić, czy nowa dolna granica na pewno będzie mniejsza niż górna granica losowania. W przeciwnym wypadku program nie będzie działał prawidłowo.

```
def sprawdzanie_poprawnosci(self):
    global b
    global a
    t = self.text1.get()
    podany_tekst = t
    try:
        liczba = int(podany_tekst)
        a = liczba
        if a > b:
            popup_showinfo("Błąd","Podana
            liczba jest większa niż górna
            granica.\nNajpierw zwiększ górną
            granicę.")
        else:
            app.l1text.set(
                "\nAby rozpocząć grę,\n wpisz
                liczbę całkowitą z przedziału
                od " + str(a)
                + " do " + str(b) + ",
                \nktóra Twoim zdaniem została
                wylosowana :\n")
            global_n_randint(a,b)
    except:
        popup_showinfo("Błąd", "Podane znaki
        to nie liczba całkowita")
    self.okno.grab_release()
    self.okno.destroy()
```

Rys. 14.8. Utworzenie klasy okna modalnego – metoda walidująca ciąg znaków wpisany przez użytkownika

Następny krok to utworzenie bardzo podobnej klasy, której zadaniem będzie obsłużenie opcji „Podaj liczbę końca przedziału losowania". Klasa wygląda analogicznie do wcześniejszej, z tym wyjątkiem, że teraz musi mieć metodę sprawdzającą, czy podana liczba na pewno jest większa od aktualnej dolnej granicy losowania.

```
class podaj_liczbe_koniec:
    def __init__(self):
        self.okno = tk.Toplevel()
        self.okno.wm_title("Podaj liczbę b")
        self.okno.grab_set()
        self.l = tk.Label(self.okno, text="Podaj
        liczbę końca przedziału losowania")
        self.l.grid(row=0, column=0)

        self.text1 = tk.Entry(self.okno)
        self.text1.grid(row=1,column=0)
        self.b = ttk.Button(self.okno, text="Ok",
        command=self.sprawdzanie_poprawnosci)
        self.b.grid(row=1, column=1)
```

Rys. 14.9. Utworzenie kolejnej klasy okna modalnego, pobierającej od użytkownika liczbę i sprawdzającej jej poprawność – konstruktor

Klasa „podaj_liczbe_poczatek" i klasa „podaj_liczbe_koniec" mają wiele wspólnych elementów. W teorii mogłyby być dla naszego programu jedną klasą, lecz w praktyce musielibyśmy:

- utworzyć bardzo rozbudowany konstruktor z kilkunastoma argumentami zapisującymi kilkanaście odpowiadających mu pól w klasie, co niepotrzebnie skomplikowałoby program;
- w przyszłości napotkać poważny problem podczas rozwijania projektu w dwóch różnych kierunkach.

Mimo że wykonują one podobne zadania, napisaliśmy dwie osobne klasy, aby utrzymać prostotę i czytelność kodu oraz zapewnić prostą modyfikację w przyszłości.

```
def sprawdzanie_poprawnosci(self):
    global b
    global a
    t = self.text1.get()
    podany_tekst = t
    try:
        liczba = int(podany_tekst)
        b = liczba
        if a > b:
            popup_showinfo("Błąd","Podana
            liczba jest mniejsza niż dolna
            granica.\nNajpierw pomniejsz
            dolną granicę.")
        else:
            app.l1text.set(
                "\nAby rozpocząć grę,\n wpisz
                liczbę całkowitą z przedziału
                od " + str(a)
                + " do " + str(b) + ",
                \nktóra Twoim zdaniem została
                wylosowana :\n")
            global_n_randint(a,b)
    except:
        popup_showinfo("Błąd", "Podane znaki
        to nie liczba całkowita")
    self.okno.grab_release()
    self.okno.destroy()
```

Rys. 14.10. Utworzenie kolejnej klasy okna modalnego – metoda walidująca ciąg znaków wpisany przez użytkownika

Kolejnym prostym elementem do utworzenia będzie funkcja (rys. 14.11).

```
def popup_showinfo(title="Window", tekst="Hello
World!"):
    showinfo(title, tekst)
```

Rys. 14.11. Aplikacja – prosta funkcja wywołująca okno modalne powiadomienia dla użytkownika

Potrzebujemy jeszcze utworzonej wcześniejszej rozszerzonej klasy Entry (rys. 14.12).

```
class EntryRozszerzone(tk.Entry):
    def __init__(self, master=None,
    tekst_zastepczy="podaj liczbe", color='grey'):
        super().__init__(master)

        self.tekst_zastepczy = tekst_zastepczy
        self.tekst_zastepczy_color = color
        self.default_fg_color = self['fg']

        self.bind("<Key>", self.focus_in)
        self.bind("<Leave>", self.focus_out)
        self.put_tekst_zastepczy()

    def put_tekst_zastepczy(self):
        self.delete('0', 'end')
        self.insert(0, self.tekst_zastepczy)
        self['fg'] = self.tekst_zastepczy_color

    def focus_in(self, *args):
        if self['fg'] == self.
        tekst_zastepczy_color:
            self.delete('0', 'end')
            self['fg'] = self.default_fg_color

    def focus_out(self, *args):
        if not self.get():
            self.put_tekst_zastepczy()
```

Rys. 14.12. Klasa EntryRozszerzone umożliwia wyświetlanie podpowiedzi dla użytkownika na elemencie Entry

Na koniec musimy utworzyć główną klasę aplikacji „Zgadnij liczbę". Zaczniemy od napisania konstruktora, a w nim – jako pierwsze – menu (rys. 14.13).

```
class Application(ttk.Frame):
    def __init__(self, master):
        self.z = tk.Frame.__init__(self, master)
        self.pack()
        menubar = tk.Menu(root) # MENU
        filemenu = tk.Menu(menubar,tearoff=0)
        filemenu.add_command(label="Zmień liczbę
        początku przedziału",
        command=podaj_liczbe_poczatek)
        filemenu.add_command(label="Zmień liczbę
        końca przedziału",
        command=podaj_liczbe_koniec)
        filemenu.add_separator()
        filemenu.add_command(label="Exit",
        command=root.quit)
        menubar.add_cascade(label="Program",
        menu=filemenu)
        helpmenu = tk.Menu(menubar, tearoff=0)
        helpmenu.add_command(label="O programie",
        command=lambda: popup_showinfo("O
        programie","Gra 'Zgadnij Liczbę'"))
        helpmenu.add_command(label="O Autorze",
        command=lambda: popup_showinfo("O
        Autorze","Autor: Dariusz Gołębiowski"))
        menubar.add_cascade(label="Pomoc",
        menu=helpmenu)
        root.config(menu=menubar)
```

Rys. 14.13. Główna klasa – menu aplikacji

Następnym krokiem jest dodanie elementów na górze okna – baneru i opisu gry (rys. 14.14).

```
# Górna część okna
self.high_frame = tk.Frame(self.z)
self.high_frame.configure
(background=bg_color)
self.high_frame.pack()

# baner
self.load = Image.open("baner3.png")
self.img = ImageTk.PhotoImage(self.load)
self.l0 = tk.Label(self.high_frame,
image=self.img, bg=bg_color)
self.l0.grid(row=0,column=0,sticky="N")

self.l001 = tk.Label(self.high_frame,
text="Witaj serdecznie.",bg=bg_color,
fg=text_color,font="none 16 bold").grid
(row=1, column=0)

self.l1text = tk.StringVar()
self.l1text.set("\nAby rozpocząć grę,\n
wpisz liczbę całkowitą z przedziału od "
+str(a)+" do "+str(b)+", \nktóra Twoim
zdaniem została wylosowana :\n")
self.l1 = tk.Label(self.high_frame,
textvariable=self.l1text,bg=bg_color,
fg=text_color,font="none 12 bold").grid
(row=2,column=0, sticky="W")
```

Rys. 14.14. Główna klasa – baner i opis aplikacji

Następnie dodajemy (rys. 14.15):

- klasę EntryRozszerzone, aby pobrać wejście od użytkownika;
- element Button, którego naciśnięcie lewym klawiszem myszy wywoła metodę **`click()`**, co spowoduje przesłanie ciągu znaków od użytkownika (do walidacji w dalszej części programu).

```
# środek aplikacji - pobranie informacji
od użytkownika

self.middle_frame = tk.Frame(self.z)
self.middle_frame.configure
(background=bg_color)
self.middle_frame.pack(padx=5)

self.tekst1 = EntryRozszerzone(self.
middle_frame, tekst_zastepczy="podaj
liczbe")
self.tekst1.grid(row=0,column=0)

tk.Label(self.middle_frame,text="",
width=5,bg=bg_color).grid(row=0,column=1)
self.b1 = tk.Button(self.middle_frame,
text="Podaj liczbę",width=9, command=self.
click).grid(row=0,column=2)
```

Rys. 14.15. Główna klasa – pobranie informacji od użytkownika i przycisk do jej walidacji

Potrzebujemy jeszcze elementu wyświetlającego informację, czy gracz zgadł liczbę bądź czy podał mniejszą lub większą liczbę, i ewentualnie wskazanie graczowi, że potrzebujemy od niego liczby, a nie tekstu (rys. 14.16).

Na rys. 14.16 zapisany obiekt „l2" zawiera obiekt StringVar, w którym przechowujemy „\n\n\n". Rezerwuje nam to trzy linie w oknie aplikacji, których będziemy potrzebowali w pewnym momencie działania aplikacji (jednak nie na początku).

```
# Wyświetlenie elementów gry
self.low_frame = tk.Frame(self.z)
self.low_frame.configure
(background=bg_color)
self.low_frame.pack()

self.l4text = tk.StringVar()
self.l4text.set("\n\n\n")
self.l2 = tk.Label(self.low_frame,
textvariable=self.l4text,bg=bg_color,
fg=text_color,font="none 16 bold").grid
(row=3,column=0,sticky="W")
```

Rys. 14.16. Główna klasa – użycie Label dla późniejszej informacji zwrotnej przekazywanej użytkownikowi

Ostatnimi widocznymi częściami interfejsu gry są: napis na dole ekranu „Naciśnij, aby opuścić program" i przycisk „Wyjście" (rys. 14.17).

```
# dolna część programu
self.lowest_frame = tk.Frame(self.z)
self.lowest_frame.configure
(background=bg_color)
self.lowest_frame.pack()
l4 = tk.Label(self.lowest_frame,
text="Naciśnij,aby opuścić program ",
bg=bg_color,fg=text_color,font="none 16
bold").grid(row=5,column=0)

tk.Button(self.lowest_frame,
text="Wyjście",width=6, command=self.
close_window).grid(row=5,column=1)
```

Rys. 14.17. Główna klasa – napis na dole ekranu i przycisk służący do zakończenia działania aplikacji

Przedostatnią częścią klasy naszej aplikacji jest metoda **`click`**, odpowiadająca za:

- walidację wejścia użytkownika;
- resetowanie tekstu zastępczego (podpowiedzi dla użytkownika, co powinien wpisać).

Kod przedstawiono na rys. 14.18.

```
def click(self):
    global a
    global b
    podany_tekst = self.tekst1.get()
    try:
        liczba = int(podany_tekst)
        if liczba == n:
            liczba = cZgadzaSie + str(liczba)
            + "\n"
            self.l4text.set(liczba)
            MsgBox = tk.messagebox.askquestion
            ('WYGRAŁEŚ','Gratulacje, czy
            chesz zagrać jeszcze raz?')
            if MsgBox == 'yes':
                global_n_randint(a,b)
            else:
                pass
        elif liczba > n:
            liczba = bLiczbaJestMniejsza
        else:
            liczba = aLiczbaJestWieksza
    except:
        liczba = "\nPodane znaki '"
        +podany_tekst+ "' to nie
        liczba\nSpróbuj ponownie.\n"
    self.l4text.set(liczba)
    self.tekst1.put_tekst_zastepczy()
```

Rys. 14.18. Główna klasa – metoda **`click()`**

Ostatnią metodą jest standardowa metoda zamykająca okno i czyszcząca pamięć podręczną programu. Przedstawiono ją na rys. rys. 14.19.

```
    def close_window(self):
        root.destroy()
        exit()


root = tk.Tk()
root.title("Zgadnij liczbę")
root.configure(background=bg_color)
app = Application(root)

root.mainloop()
```

Rys. 14.19. Główna klasa – metoda **`close_window()`** i start programu

Pod metodą zawarto kod poprawnego uruchomienia aplikacji.

14.3. Eksport aplikacji do pliku .exe

Aby udostępnić grę innemu komputerowi z systemem Windows, który nie ma skonfigurowanego środowiska uruchomieniowego języka Python, trzeba utworzyć plik wykonywalny .exe. Będziemy do tego potrzebowali biblioteki Pythona „pyinstaller". Aby ją zainstalować, wpisz w swoją konsolę poniższą instrukcję:

```
pip install pyinstaller
```

Następnie przejdź w konsoli za pomocą polecenia „cd" do folderu, w którym znajduje się plik aplikacji, i wpisz poniższą instrukcję:

```
pyinstaller.exe --onefile main.py
```

Powinieneś zobaczyć w tym folderze utworzony plik main.exe. Gra wykonana we frameworku tkinter została ukończona.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wyjaśnij pojęcia okno modalne i okno niemodalne?
2. Wyjaśnij następujące instrukcje:
   a) `import tkinter as tk`,
   b) `from tkinter import ttk`,
   c) `from tkinter.messagebox import showinfo`,
   d) `import random`,
   e) `from PIL import Image, ImageTk`.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. W kodzie powyższej aplikacji jest funkcja **`popup_showinfo`**. Odszukaj ją. Wywołuje ona z tkinter.messagebox funkcję **`showinfo`**. W tkinter.messagebox są również funkcje **`showwarning`** i **`showerror`**. Działają one tak samo jak funkcja **`showinfo`**, tyle że zamiast rysunku informacji w oknie znajduje się rysunek ostrzeżenia i błędu (dokładny wygląd tego obrazka zależy od danego systemu operacyjnego). Twoim zadaniem jest dodanie dodatkowego parametru do funkcji **`popup_showinfo – numer_wywołania`**, który będzie liczbą od 0 do 2, a także gdy funkcja zostanie wywołana z numerem:
   a) 0 – zostanie wywołana **`showinfo`**;
   b) 1 – zostanie wywołana **`showwarning`**;
   c) 2 – zostanie wywołana **`showerror`**.

   Zmień w aplikacji wywołania funkcji **`popup_showinfo`** w taki sposób, że jeżeli użytkownik poda:
   a) nie liczbę, to wyskoczy okno showerror;
   b) liczbę (podczas wyboru nowego zakresu górnego) mniejszą niż aktualny zakres dolny, powinno wyskoczyć okno showwarning;
   c) liczbę (podczas wyboru nowego zakresu dolnego) większą niż aktualny zakres górny, powinno wyskoczyć okno showwarning.

   Jeżeli żaden z powyższych warunków nie zostanie spełniony, powinno wyskoczyć okno **`showinfo`**.

2. Pod opisem gry dodaj na ekranie głównym aplikacji:
   - obiekt EntryRozszerzone z Buttonem, w którym użytkownik może zmienić w dowolnym momencie gry zakres dolny losowania liczby – analogiczne do opcji z menu „Zmień liczbę początku losowania”; z tym wyjątkiem, że ma działać bez pierwszego wyskakującego okna – jak zgadywanie liczby przez użytkownika;
   - obiekt EntryRozszerzone z buttonem, w którym użytkownik może zmienić w dowolnym momencie gry zakres górny losowania liczby – analogiczne do opcji z menu „Zmień liczbę końca losowania”; z tym wyjątkiem, że ma działać bez pierwszego wyskakującego okna – jak zgadywanie liczby przez użytkownika.

3. Z użyciem frameworku języka programowanie Python tkinter napisz zaawansowaną wersję gry „Papier – kamień – nożyce".
   Gra ma wyglądać następująco:
   1. Na górze powinien być obiekt typu Label opisujący grę.
   2. Poniżej powinien znajdować się Label informujący użytkownika, że może wybrać jeden z trzech poziomów trudności (wpisując słowo bądź liczbę do poniższego obiektu typu EntryRozszerzone).
   3. Element EntryRozszerzone z odpowiadającym mu przyciskiem – obiektem typu Button.
   4. Label z opisem, ile punktów ma użytkownik, a ile – komputer.
   5. Drugi element typu EntryRozszerzone, na wejście wyboru użytkownika: „Papier – kamień – nożyce" i odpowiadający mu obiekt Button walidujący wejście użytkownika.
   6. Label z opisem, kto wygrał grę.

   Działanie gry:
   1. Jeżeli w pierwszym obiekcie typu EntryRozszerzone użytkownik wpisał:
      a) łatwy lub 0 – komputer zawsze powinien wybrać opcję, aby przegrać, czyli jeżeli użytkownik wybierze „kamień", to komputer powinien wybrać opcję „nożyce";
      b) średni lub 1 – komputer losowo wybiera indeks z tablicy [„papier", „kamień", „nożyce"] z użyciem biblioteki random i funkcji **`random.randint(dolny_zakres_losowania, górny_zakres_losowania)`**;
      c) trudny lub 2 – komputer, wiedząc, co wybrał gracz, natychmiast podejmuje akcję, w której zawsze wybiera zwycięską opcję – czyli jeżeli użytkownik wybiera opcję „kamień", to komputer powinien wybrać opcję „papier";
      d) jeżeli użytkownik wpisał coś innego, to powinno się pojawić okno modalne showinfo i powiadomić użytkownika, co powinien wpisać, aby program zadziałał poprawnie.
   2. Jeżeli w drugim obiekcie typu EntryRozszerzone użytkownik wpisze cokolwiek, to po naciśnięciu przycisku Button wywołuje się metoda, która:
      a) przepisuje tekst do zwykłego obiektu typu string z użyciem konstrukcji:

      ```
      nowy = self.obiekt_varstring.get()
      ```

      Następnie – dla ułatwienia przyrównań – zamienia duże litery na małe z użyciem metody **`lower()`**:

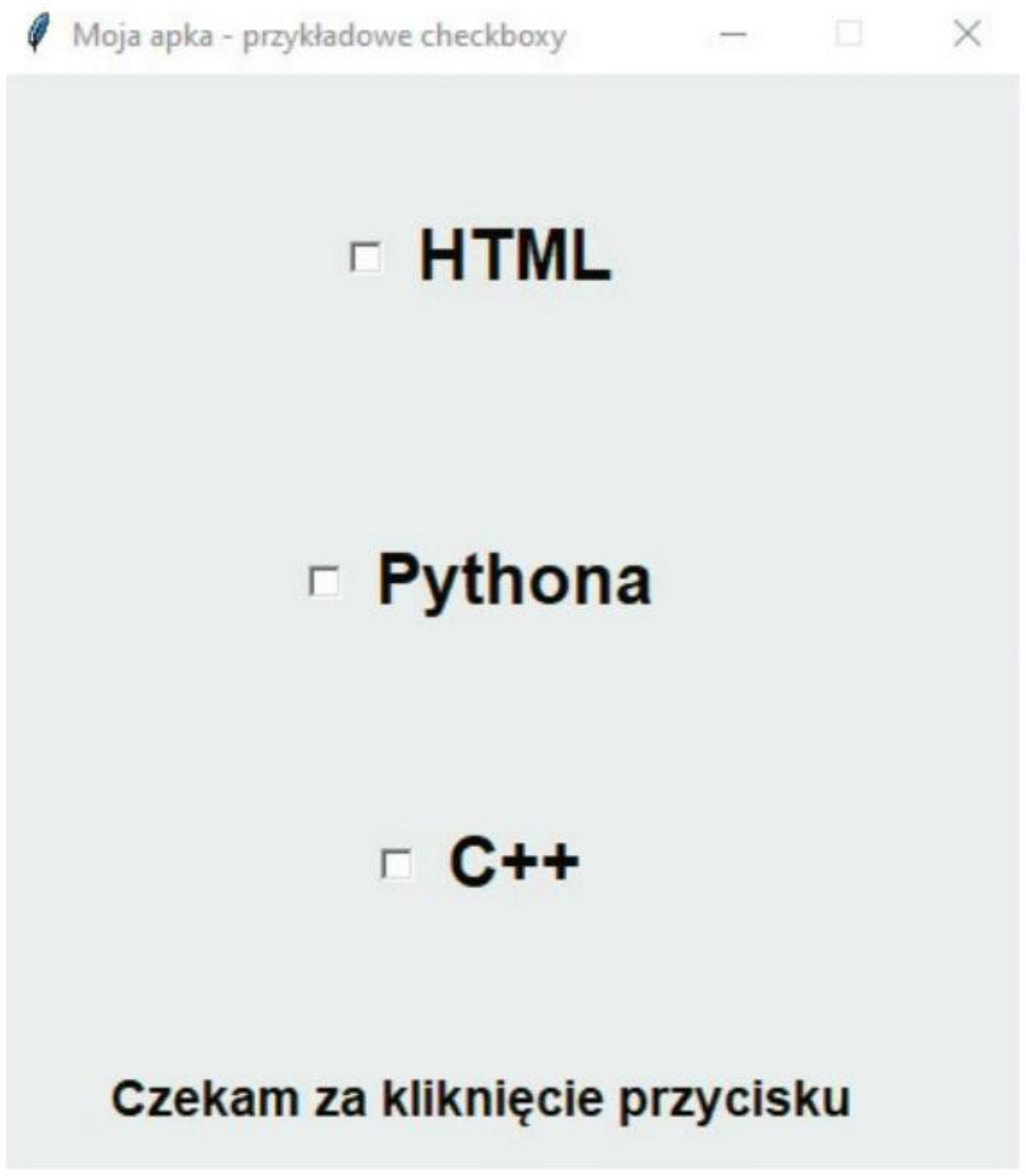
      ```
      nowy = nowy.lower()
      ```

      b) sprawdza, czy tekst podany przez użytkownika jest równy:
      - „papier" – według zasad z punktu pierwszego wykonuje logikę poziomu trudności gry;
      - „kamień" – według zasad z punktu pierwszego wykonuje logikę poziomu trudności gry;
      - „nożyce" – według zasad z punktu pierwszego wykonuje logikę poziomu trudności gry;
      - wyświetla informację o błędnym wejściu użytkownika z użyciem okna modalnego „showinfo".

3. Jeżeli któryś z graczy zdobył 3 punkty przewagi lub 10 punktów jako pierwszy – wygrywa grę. Wówczas powinno się pojawić okno modalne z użyciem „showinfo", które wypisuje, kto wygrał. Po zamknięciu okna punkty gracza i komputera powinny się wyzerować na ekranie.
4. Po każdej zmianie trudności gry wyniki komputera i użytkownika powinny się zerować.
5. Błędne wejście użytkownika do któregokolwiek z obiektów EntryRozszerzone nie zeruje gry.

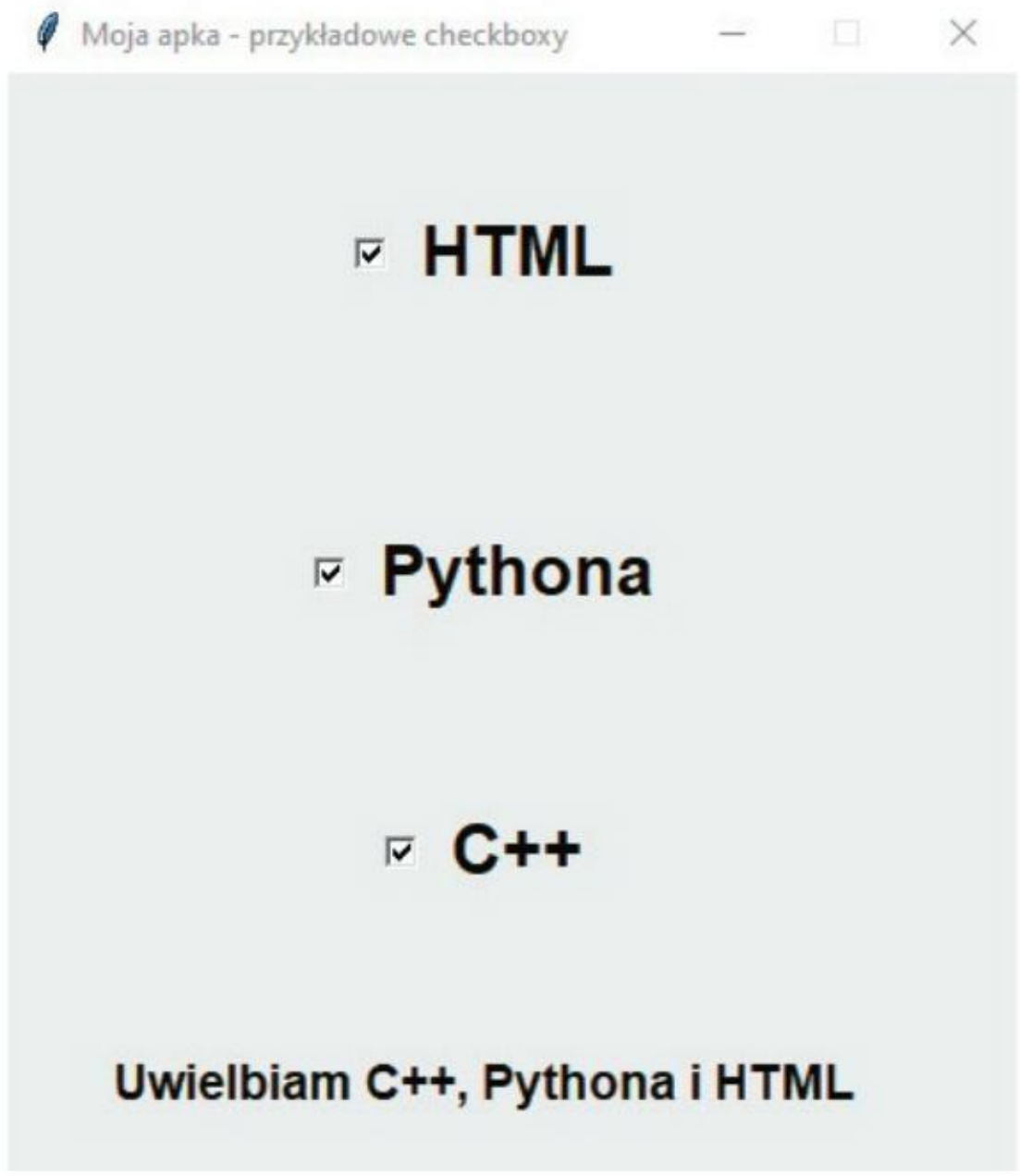
4. Stwórz okno aplikacji, które:
   a) ma rozmiar 400 pikseli na 500 pikseli,
   b) ma tytuł „Przykładowy obrazek".
   Dodaj wewnątrz obrazek z użyciem frameworku tkinter i biblioteki PIL (pillow) – tak jak było to opisane w kodzie naszej aplikacji „Zgadnij liczbę",
   Dodaj element Label pod zdjęciem.

5. Za pomocą elementu Checkbutton stwórz aplikację jak na rys. 14.20.



Rys. 14.20. Aplikacja z elementami typu checkbox

Obiekty Checkbutton zachowują się jak elementy HTML-a checkbox. Twoim zadaniem będzie napisanie takiej metody, aby po naciśnięciu danego Checkbuttonu zmieniał się text w elemencie Label. Przykładowe zaznaczenie wszystkich przycisków powinno dać efekt jak na rys 14.21.



Rys. 14.21. Aplikacja z elementami typu checkbox – przykład wciśnięcia wszystkich Checkbuttonów

Checkbutton jest połączeniem obiektu typu Label i obiektu typu Button, ponieważ:

- ma odpowiadający element przechowujący informację o jego stanie – **`tk.IntVar()`** (tak jak był **`tk.StringVar()`** dla Labeli);
- w konstruktorze ma argument **`command`** – do wywoływania funkcji / metod.

Przykład utworzenia obiektu Checkbutton wygląda następująco:

```
self.c1 = tk.Checkbutton(okno, text="HTML", variable=self.
obiekt_IntVar, onvalue=1, offvalue=0, command=lambda: self.
zmien_tekst(self.label01,"HTML"), width=20, font="none 20
bold"),
```

gdzie:

- **`text`** – tekst wyświetlany obok okienka do potwierdzenia opcji;
- **`variable`** – przechowuje obiekt, który pamięta wartość liczbową elementu;
- **`onvalue`** – określenie, jaka wartość zostanie zapisana do „variable” po kliknięciu elementu;
- **`offvalue`** – określenie, jaka wartość zostanie zapisana do „variable” po odkliknięciu elementu;
- **`width`** – szerokość elementu;
- **`font`** – czcionka, jaką zostanie wydrukowany element „text”.

IV. Programowanie aplikacji mobilnych

- **Wykorzystanie środowiska programistycznego dla aplikacji mobilnych na podstawie języka Python**
- **Programowanie aplikacji mobilnych opartych na języku programowania Python**

15 Wykorzystanie środowiska programistycznego dla aplikacji mobilnych na podstawie języka Python

ZAGADNIENIA

- Stworzenie środowiska wirtualnego
- Pierwsza aplikacja mobilna
- Przygotowanie pakietów
- Alternatywne technologie tworzenia aplikacji mobilnych

15.1. Wprowadzenie

Programista może stworzyć programy użytkowe, gry, poradniki, twórcze i popularne apki. Aby osiągnąć sukces w zawodzie, potrzebna jest rozległa wiedza, związana m.in. z:

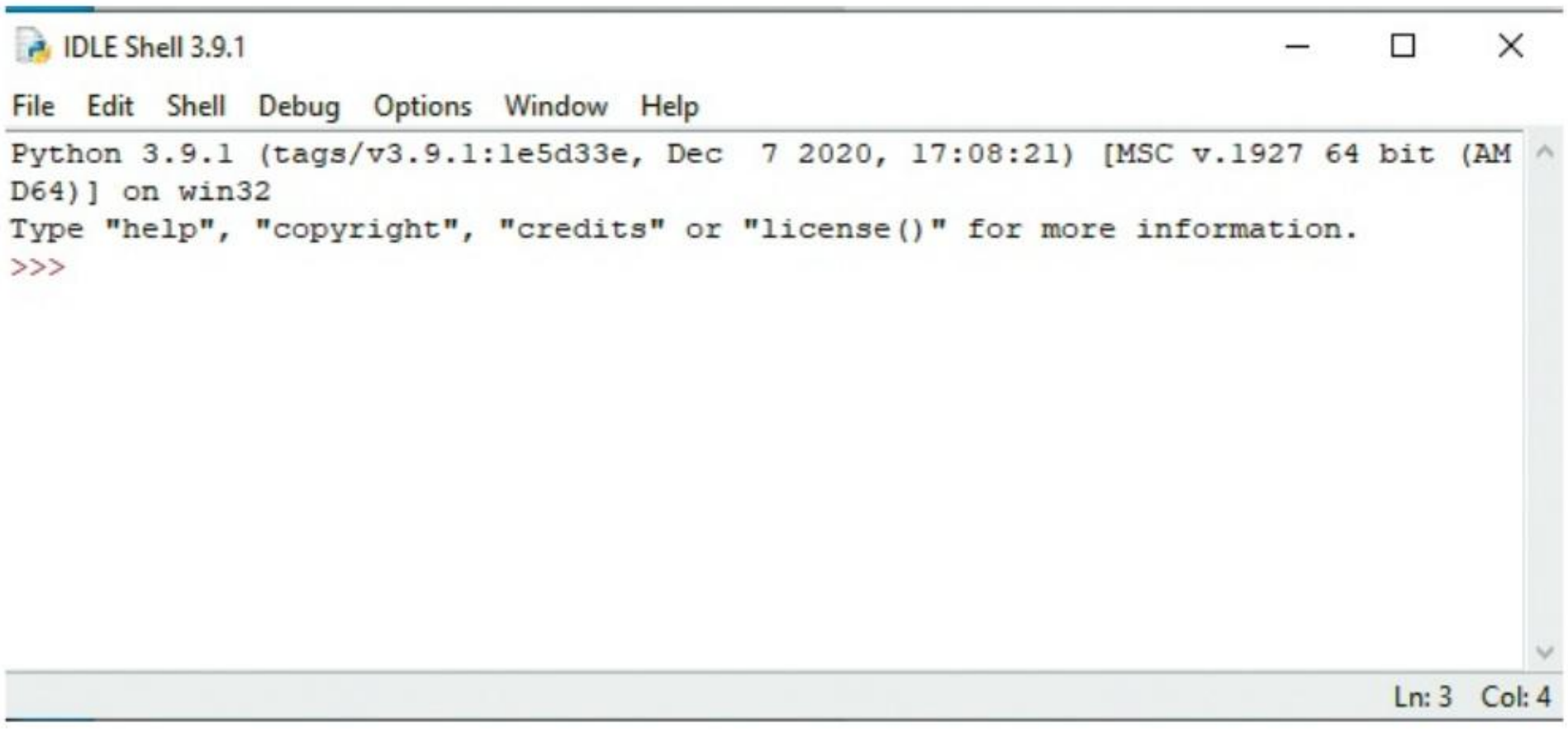
- planowaniem i projektowaniem aplikacji;
- umiejętnościami graficznymi, w tym także obsługą wybranego programu do grafiki cyfrowej;
- szeroką wiedzą informatyczną, obejmującą znajomość funkcjonowania systemów operacyjnych;
- kompetencjami programistycznymi – głównie obejmującymi programowanie w Pythonie (w przypadku technologii przedstawianej w tym podręczniku), przede wszystkim programowanie obiektowe.



Rys. 15.1. Instalacja języka Python 3.9.1

Aby sprawnie zrealizować zadania w tym rozdziale, musisz:

- mieć podstawową wiedzę dotyczącą programowania w języku Python;
- znać podstawowe komendy dotyczące konsoli w systemach operacyjnych:
  - Linux, terminal, język skryptowy BASH,
  - Windows, CMD (ewentualnie PowerShell) i jego język skryptowy BATCH;
- mieć zainstalowane w systemie operacyjnym Pythona 3+ i menedżera pakietów pip.



Rys. 15.2. Konsola języka programowania Python

Budowę aplikacji mobilnych oprzemy na języku programowania Python 3+ i frameworku Kivy. Warto zajrzeć na stronę domową opisywanego projektu – https://kivy.org/. Można tam znaleźć wiele wiadomości o frameworku Kivy, w tym różnego rodzaju poradniki i bardzo dobrze przygotowaną dokumentację projektu. Na wskazanej stronie zamieszczono bardzo istotną informację:

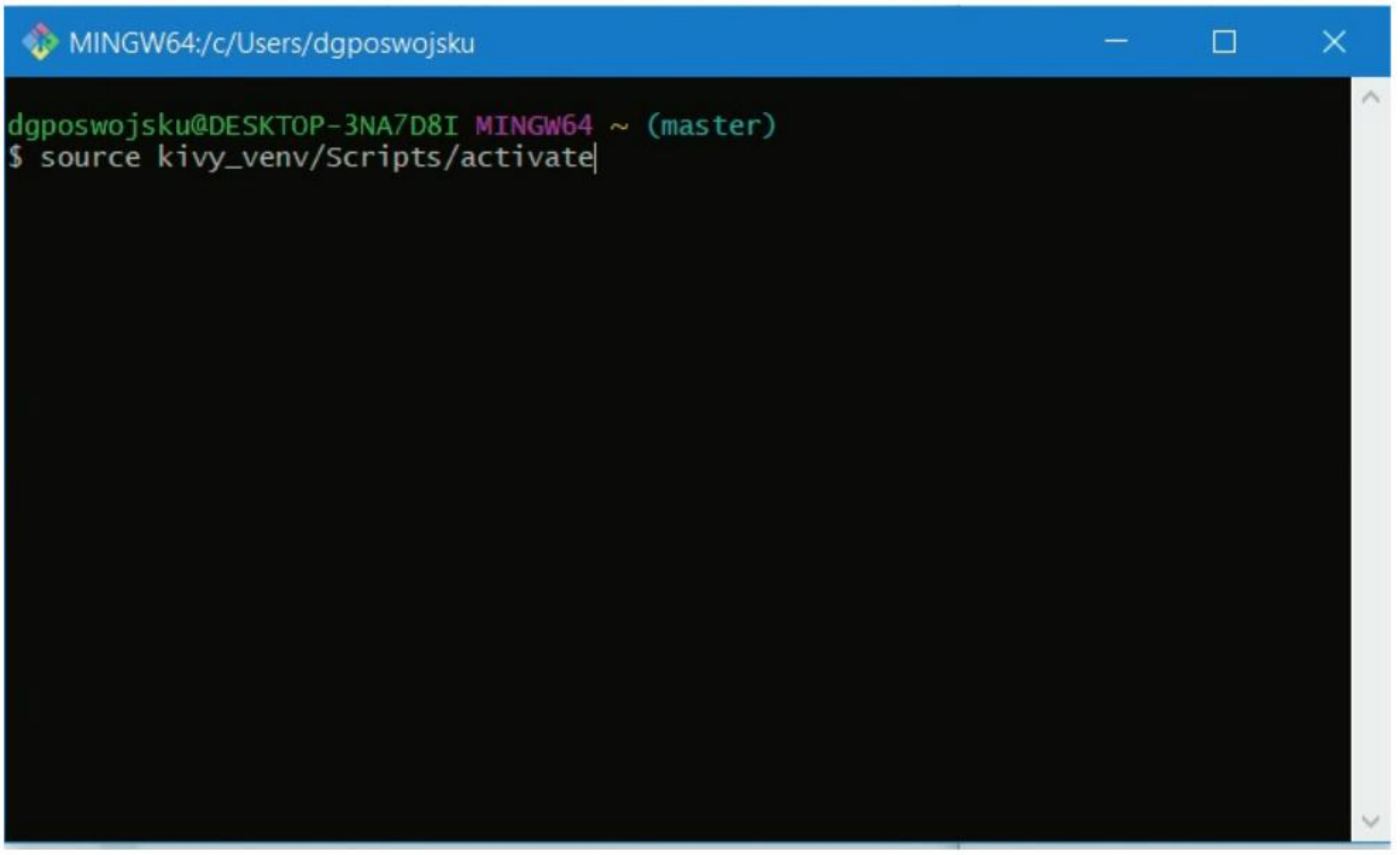
„Kivy is 100% free to use, under an MIT license
(starting from 1.7.2) and LGPL 3 for the previous versions".

Kivy jest projektem bezpłatnym w użytkowaniu, a licencja, na której jest rozpowszechniany – MIT (szczegóły licencji: https://opensource.org/licenses/MIT), pozwala, w przypadku zachowania poszanowania dla twórców projektu, wykorzystywać go zarówno w projektach komercyjnych, jak i prywatnych. Warto wspomnieć, że Kivy współpracuje z Pythonem w wersjach od 3.6 do 3.9. Co ważne, w momencie tworzenia swojej aplikacji będziesz musiał wejść na stronę projektu i upewnić się, której wersji wymienionego języka programowania trzeba użyć.

Brak opłat związanych bezpośrednio z Kivy nie oznacza, że wszystkie projekty w nim zrobione można dowolnie dystrybuować. Warunek jest taki, że będą działały w środowisku Python. W momencie korzystania z eksportu projektów do plików umożliwiających ich działanie bezpośrednio w danym systemie operacyjnym trzeba zwrócić uwagę na ograniczenia licencyjne, należące do właścicieli tych systemów. Temat jest bardzo obszerny i dość skomplikowany pod względem prawnym, więc gdy podejmiesz decyzję

o upowszechnieniu aplikacji, będziesz musiał przeanalizować wszystkie aspekty prawne. Wsparcie znajdziesz na stronach projektu Kivy: https://kivy.org/doc/stable/guide/licensing.html.

Projekt Kivy współpracuje ze wszystkimi najpopularniejszymi systemami operacyjnym. Skupimy się na instalacji i obsłudze w dwóch najpopularniejszych systemach: Linux (dystrybucje oparte na Debianie – źródło: https://www.debian.org/index.en.html) i Windows 10 (źródło: https://www.microsoft.com/pl-pl/software-download/windows10). W innych systemach sposób postępowania jest podobny, zmienią się jedynie szczegóły, które możesz poznać na stronie omawianego frameworka: https://kivy.org/doc/stable/gettingstarted/installation.html.



Rys. 15.3. Konsola GIT – przykład kodu

Zanim przystąpimy do programowania, warto wiedzieć, co jeszcze można zrobić za pomocą frameworka Kivy. Najważniejsza jest aplikacja, w pełni działająca, sprawna, funkcjonalna apka, ale funkcjonująca jedynie w obszarze środowiska Python.

Po utworzeniu aplikacji trzeba przygotować pakiet dla danego systemu operacyjnego. Omawiana technologia umożliwia przygotowanie kilku wersji paczek, m.in. na systemy:

- Android,
- Windows,
- OS X,
- iOS.

Zanim zdecydujesz się na stworzenie pakietu na dany system, a szczególnie na jego powszechne udostępnienie, warto przemyśleć zakres praw autorskich. Najbardziej liberalne podejście do prawa licencyjnego można zaobserwować w przypadku Androida. Windows

jest dużo bardziej rygorystyczny, a OS X i iOS są znane z bardzo konserwatywnych zasad licencjonowania wszystkiego, co jest z nimi związane. Trzeba uważanie czytać zapisy licencyjne (dotyczące prawa autorskiego na system i jego elementy).

15.2. Stworzenie środowiska wirtualnego

Tworząc oprogramowanie użytkowe, powinniśmy zawsze zadbać o to, aby nasze środowisko programistyczne było aktualne (w miarę możliwości i potrzeb). Zaczniemy od aktualizacji instalacji Pythona. Aby to wykonać, otwórz:

- system Windows – konsolę CMD (ewentualnie PowerShell lub Bash);
- system Linux – terminal.

```
C:\>mkdir kivy

C:\>cd kivy

C:\kivy>
```

Rys. 15.4. Stworzenie folderu w CMD

Następnie w konsoli / terminalu wpisz polecenie i je uruchom:

```
python -m pip install --upgrade pip setuptools virtualenv
```

```
PS C:\> cd kivy
PS C:\kivy>
PS C:\kivy> python -m pip install --upgrade pip setuptools virtualenv
Requirement already satisfied: pip in c:\python38\lib\site-packages (21.0.1)
Requirement already satisfied: setuptools in c:\python38\lib\site-packages (53
.0.0)
Requirement already satisfied: virtualenv in c:\python38\lib\site-packages (20
.4.2)
Requirement already satisfied: distlib<1,>=0.3.1 in c:\python38\lib\site-packa
ges (from virtualenv) (0.3.1)
Requirement already satisfied: six<2,>=1.9.0 in c:\python38\lib\site-packages
(from virtualenv) (1.15.0)
Requirement already satisfied: appdirs<2,>=1.4.3 in c:\python38\lib\site-packa
ges (from virtualenv) (1.4.4)
Requirement already satisfied: filelock<4,>=3.0.0 in c:\python38\lib\site-pack
ages (from virtualenv) (3.0.12)
PS C:\kivy>
```

Rys. 15.5. Aktualizacja środowiska Python

W twojej konsoli mogą się pokazać inne napisy niż na rys. 15.5. Wszystko zależy od tego, co masz zainstalowane na komputerze i w systemie operacyjnym.

Przystępujemy do zainstalowania Kivy i stworzenia wirtualnego środowiska, w którym będziemy mogli oglądać pracę.

ETAP 1 – przygotowanie miejsca na aplikację

Na początek przygotujemy na twardym dysku miejsce na aplikację wykonaną w Kivy. Należy utworzyć folder mobileapp. W tym celu w konsoli / terminalu wpisz polecenie:

```
mkdir mobileapp
```

Następnie wejdź do stworzonego katalogu i wpisz:

```
cd mobileapp
```

ETAP 2 – stworzenie wirtualnego środowiska

Będąc wewnątrz katalogu mobileapp, stwórz wirtualne środowisko (wewnątrz katalogu kivy):

```
python -m virtualenv testApp
```

W kolejnym kroku aktywuj wirtualne środowisko poleceniem – dla systemu:

- Linux

```
source testApp/bin/activate
```

- Windows CMD

```
source testApp/bin/activate
```

- Windows Terminal Bash

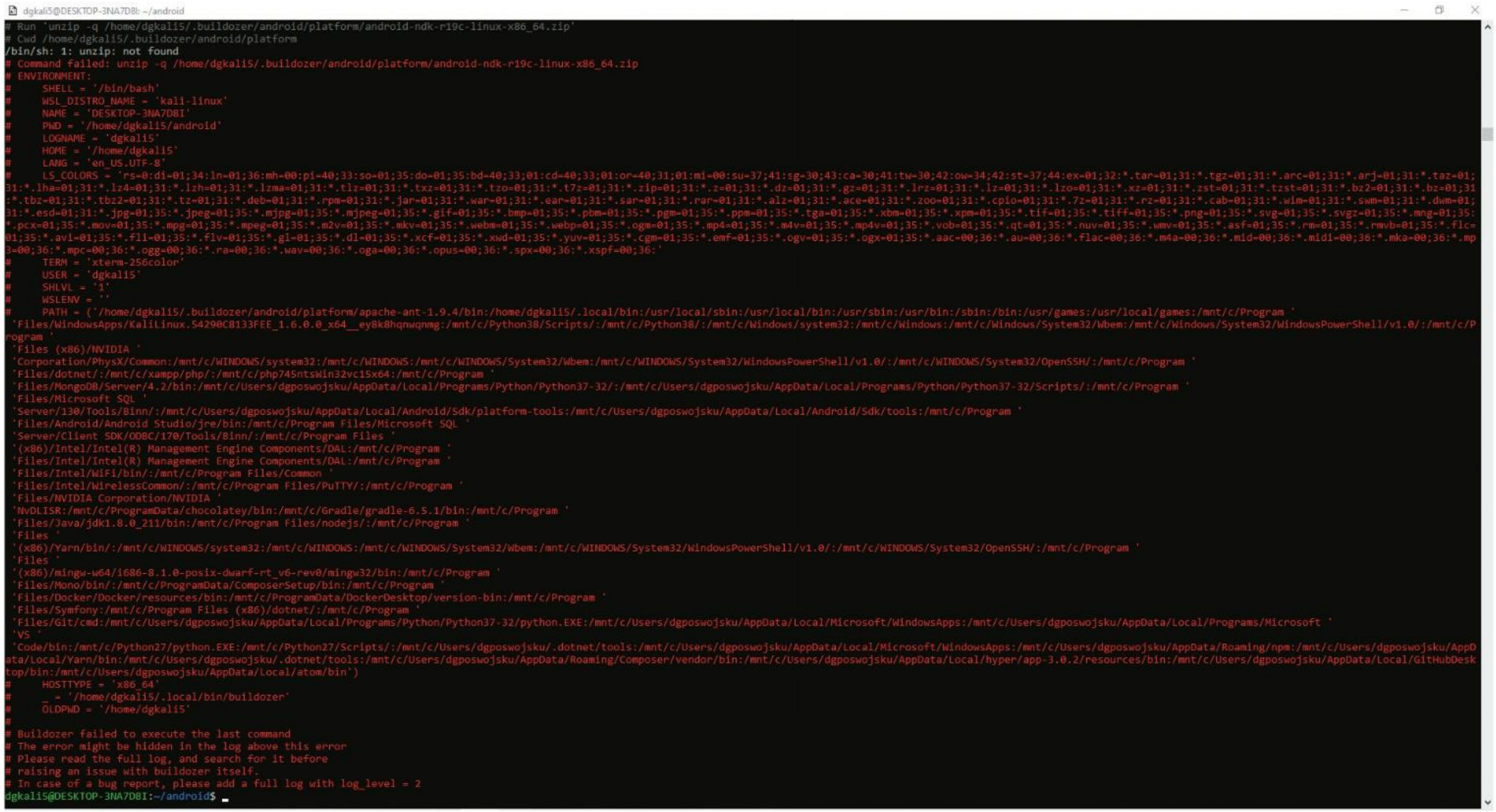
```
source testApp /Scripts/activate
```

CIEKAWOSTKA

Programiści powinni być wyczuleni na podkreślenia w kodzie, które są wyświetlane przez oprogramowanie i w związku z działaniem jego użytkowników. Zwykle mamy do czynienia z dwoma kolorami:

- pomarańczowy – ostrzega o mogących wystąpić błędach; np. gdy używamy przestarzałych struktur programistycznych, dowiadujemy się, że teraz ten kod jeszcze zadziała, ale w następnej wersji oprogramowania będzie już powodował błędy;
- czerwony – pojawiające się w tym kolorze elementy zazwyczaj wskazują na błędy, czyli stworzony kod nie będzie działał poprawnie i trzeba go poprawić.

Na rys. 15.6 widać przykład błędu powstałego w systemie operacyjnym Linux Ubuntu. Czerwony kolor napisów w terminalu ostrzega i informuje o tym, że w trakcie przetwarzania programu powstały błędy.



Rys. 15.6. Terminal w Linux – przykład błędu w wykonaniu polecenia

ETAP 3 – instalacja Kivy

Instalacja (będąc w wirtualnym środowisku) tego frameworka może obejmować:

- Kivy – wówczas aby pozyskać minimalne zależności do działania aplikacji, wystarczy poniższy fragment kodu:

  ```
  python -m pip install kivy[base]
  ```

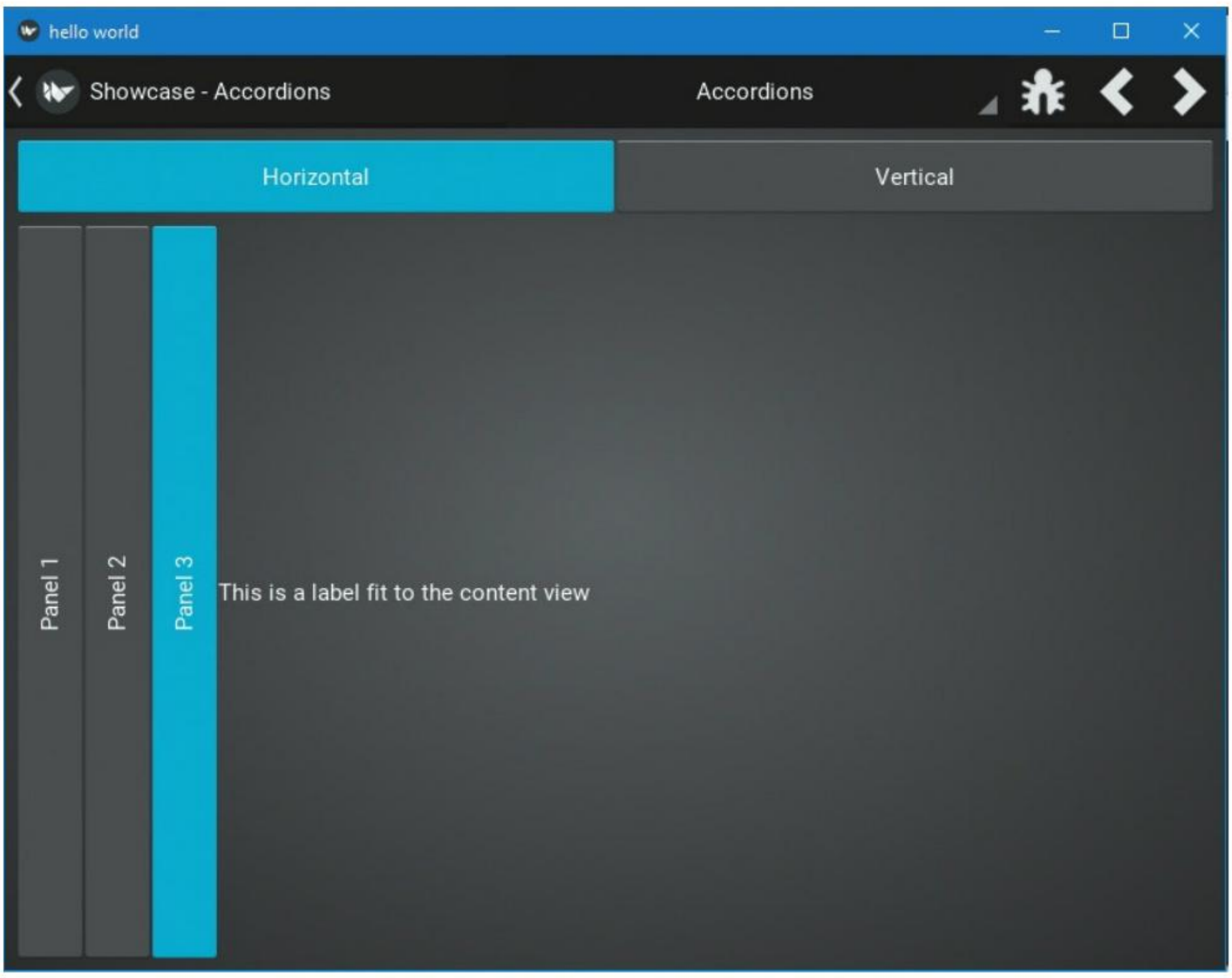
  W miejsce sformułowania **[base]** możesz także wpisać **[base,media]** lub **[full]**. Chodzi o zakres aplikacji, a ściślej – o wsparcie dla audio i video;
- Kivy i przygotowane przez zespół deweloperów tworzących ten framework kivy_examples, czyli przykładowe elementy kodu wraz z wizualizacją:

  ```
  python -m pip install kivy[base] kivy_examples
  ```

  Jeżeli instalacja została przeprowadzona zgodnie z podanymi przykładami, to możesz sprawdzić uzyskaną zawartość (demo możliwości Kivy) przez wpisanie kodu:
- Windows

  ```
  python testApp\share\kivy-examples\demo\showcase\main.py
  ```
- Linux

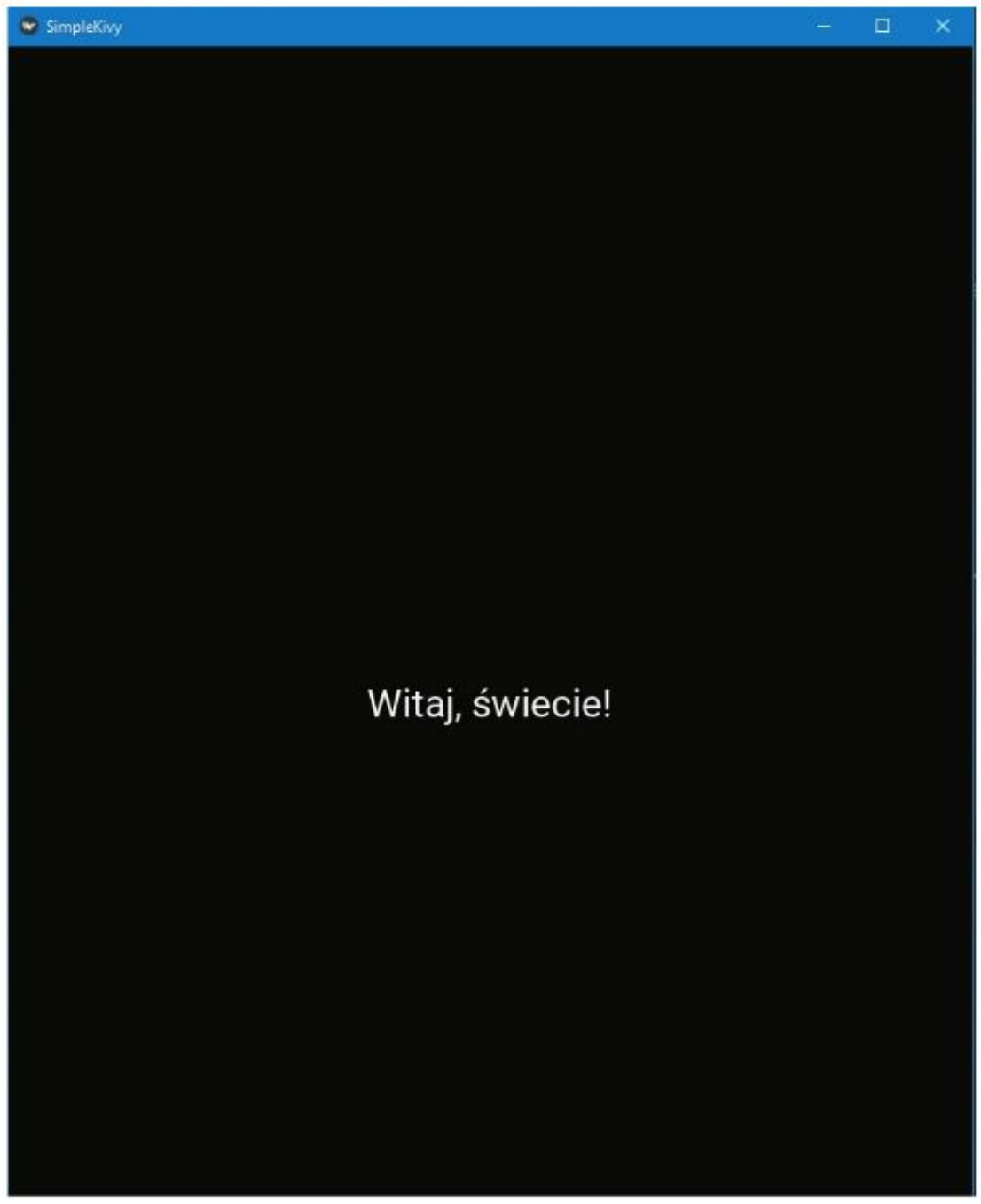
  ```
  python testApp/share/kivy-examples/demo/showcase/main.py
  ```



Rys. 15.7. Demo możliwości Kivy – efekt uruchomienia kodu po zainstalowaniu w wersji **python -m pip install kivy[base] kivy_examples**

15.3. Pierwsza aplikacja mobilna

W tej części podręcznika stworzymy prostą aplikację, która będzie wyświetlała napis „Witaj, świecie!", jak na poniższym obrazku.



Rys. 15.8. Pierwsza prosta aplikacja wykonana w Kivy – widok po jej uruchomieniu

Aby uzyskać efekt przedstawiony na rys. 15.8, stwórz w katalogu Kivy plik simpleKivy.py (w wybranym narzędziu edycyjnym lub IDE). Do wnętrza pliku wpisz kod, który utworzy wspomnianą prostą aplikację. Szczegóły – kod i opis jego działania – przedstawiono na rys. 15.9.

Dzięki rozszerzeniu pliku `.py` Windows i Python będą wiedziały, że taki plik należy uruchomić w środowisku języka programowania Python. Oczywiście stanie się tak, jeśli wcześniej wskazane środowisko zostało poprawnie zainstalowane i skonfigurowane w systemie operacyjnym. Zwykle oznacza to istnienie ścieżki dostępu do języka Python w zmiennej systemowej PATH (znajdziesz ją w „Zmiennych środowiskowych").

Podczas prac nad kodami umieszczonymi w tym podręczniku w zdecydowanej większości przypadków wykorzystywano jeden z dwóch programów:

- Visual Studio Code (strona projektu: https://code.visualstudio.com/),
- Atom (strona projektu: https://atom.io/).

Są to programy do edycji kodów programistycznych. Po dodaniu do nich kilku przydatnych modułów stają się jednak wysokiej klasy zintegrowanymi środowiskami programistycznymi (IDE).

```
import kivy
from kivy.app import App
from kivy.uix.label import Label
# Pierwsze trzy linijki importują ustawienie Kivy,
odpowiadają za pobranie odpowiednich zależności do
poprawnego działania poniższego kodu

# Definicja klasy
class SimpleKivyApp(App):

    # Funkcja, która zwróci tzw. root widget
    def build(self):

        # Etykieta z tekstem Hello World zwrócona
        zostanie jako widget główny
        return Label(text ="Witaj Świecie !")


# Inicjalizacja klasy i wywołanie metody run ()
# Czyli inicjalizuje i uruchamia się nasza
aplikacja Kivy.
SimpleKivyApp().run()
```

Rys. 15.9. Prosta aplikacja wykonana w Kivy – widok kodu wraz z opisem działania poszczególnych jego części

Kod z rys. 15.9 możesz wywołać przez wpisanie do konsoli / terminala polecenia:

```
python simpleKivy.py
```

UWAGA

Kod zadziała (otworzy się aplikacja) pod warunkiem, że środowisko Pythona zostało poprawnie zainstalowane.

Warto jeszcze wyjaśnić różnicę między tym, co przygotowujemy z wykorzystaniem frameworka Kivy, a uruchomieniem aplikacji na systemach operacyjnych. Zauważ, że na razie jednoplikowa aplikacja działa tylko wewnątrz środowiska języka Python. Oznacza to, że wszędzie tam, gdzie jest Python, jest szansa na jej uruchomienie. Co ważne, Pythona można zainstalować właściwie prawie wszędzie. Nawet bez eksportu do plików danego systemu operacyjnego można stworzyć w Pythonie projekt i uruchomić go w dowolnym systemie operacyjnym.

Warto jednak stworzyć aplikację, która będzie działała również w środowisku pozbawionym Pythona, czyli bezpośrednio w danym systemie operacyjnym. W podręczniku omówimy przygotowanie **builda**, czyli pakietu plików, który będzie można uruchomić na systemie Android, przeznaczonym dla urządzeń przenośnych. W tym zakresie Kivy będzie pomocne dzięki wykorzystaniu specjalnie do tego przygotowanego systemu buildowania.

15.4. Przygotowanie pakietów

Zajmiemy się przygotowaniem pakietów dystrybucyjnych na system Android. Szczegółowe wyjaśnienia możesz znaleźć na stronie: https://kivy.org/doc/stable/guide/packaging-android.html#packaging-android.

Pamiętaj, że technologia ciągle się zmienia i w chwili korzystania z podręcznika opisy mogą być już częściowo nieaktualne.

Pakowanie aplikacji – narzędzie PyInstaller

PyInstaller to program, który zamienia pakiety aplikacji Python w samodzielne pliki wykonywalne pod systemami:

- Android lub Windows,
- lub GNU/Linux,
- lub Mac OS X,
- lub FreeBSD,
- lub Solaris,
- lub AIX.

Powtórzenie słowa „lub" jest zamierzone, aby zrozumieć, że nie można równocześnie dokonać buildowania na wszystkie systemy, tylko trzeba przeprowadzić cały proces na każdy z nich oddzielnie. Narzędzie PyInstaller ma wiele różnorodnych opcji. Możesz je poznać na stronie https://www.pyinstaller.org/.

Utwórz folder (w terminalu poleceniem **mkdir**), w którym zostanie utworzona spakowana aplikacja, np.:

```
mkdir testApp
```

Przejdź do stworzonego katalogu:

```
cd testApp
```

Będąc wewnątrz folderu testApp, umieszczamy tam plik simpleKivy.py i wykonujemy polecenie:

```
python -m PyInstaller --name simpleKivy simpleKivy.py
```

```
C:\kivy>mkdir testApp

C:\kivy>cd testAPp

C:\kivy\testApp>python -m PyInstaller --name
simpleKivy simpleKivy.py
```

Rys. 15.10. Uruchomienie narzędzia PyInstaller

```
22013 INFO: Building PYZ (ZlibArchive) C:\kivy\testApp\build\simpleKivy\PYZ-00.p
yz
23490 INFO: Building PYZ (ZlibArchive) C:\kivy\testApp\build\simpleKivy\PYZ-00.p
yz completed successfully.
23517 INFO: checking PKG
23517 INFO: Building PKG because PKG-00.toc is non existent
23518 INFO: Building PKG (CArchive) PKG-00.pkg
23587 INFO: Building PKG (CArchive) PKG-00.pkg completed successfully.
23589 INFO: Bootloader C:\Python38\lib\site-packages\PyInstaller\bootloader\Wind
ows-64bit\run.exe
23590 INFO: checking EXE
23593 INFO: Building EXE because EXE-00.toc is non existent
23594 INFO: Building EXE from EXE-00.toc
23682 INFO: Copying icons from ['C:\\Python38\\lib\\site-packages\\PyInstaller\\
bootloader\\images\\icon-console.ico']
23756 INFO: Writing RT_GROUP_ICON 0 resource with 104 bytes
23757 INFO: Writing RT_ICON 1 resource with 3752 bytes
23758 INFO: Writing RT_ICON 2 resource with 2216 bytes
23760 INFO: Writing RT_ICON 3 resource with 1384 bytes
23761 INFO: Writing RT_ICON 4 resource with 37019 bytes
23761 INFO: Writing RT_ICON 5 resource with 9640 bytes
23761 INFO: Writing RT_ICON 6 resource with 4264 bytes
23762 INFO: Writing RT_ICON 7 resource with 1128 bytes
23766 INFO: Appending archive to EXE C:\kivy\testApp\build\simpleKivy\simpleKivy
.exe
23863 INFO: Building EXE from EXE-00.toc completed successfully.
23867 INFO: checking COLLECT
23867 INFO: Building COLLECT because COLLECT-00.toc is non existent
23868 INFO: Building COLLECT COLLECT-00.toc
26222 INFO: Building COLLECT COLLECT-00.toc completed successfully.
```

Rys. 15.11. Efekt końcowy uruchomienia narzędzia PyInstaller

Tworzenie pakietów dla Androida

Sposobów na utworzenie pakietu dla systemu operacyjnego Android jest kilka, np.:

- za pomocą narzędzia Buildozer; szczegóły znajdziesz na stronie projektu – https://github.com/kivy/buildozer;
- za pomocą pakowania z python-for-android; szczegóły znajdziesz na stronie projektu – https://python-for-android.readthedocs.io/en/latest/quickstart/;
- za pomocą pakowania aplikacji dla Kivy Launcher; szczegóły znajdziesz na stronie projektu – https://kivy.org/doc/stable/guide/packaging-android.html#packaging-your-application-into-apk.

Narzędzie Buildozer pozwala na automatyzację procesu, natomiast python-for-android wymaga nieco bardziej zaawansowanej wiedzy. Na podanej powyżej stronie znajdziesz kompleksowy opis wszystkich czynności programistycznych, które trzeba wykonać.

W dalszych zasobach tej części podręcznika znajdziesz informacje o tworzeniu wirtualnego środowiska. W tworzonej przez nas aplikacji (w systemie operacyjnym Windows – konsola standardowa CMD) korzystamy ze standardowego polecenia (rys. 15.12):

```
python -m virtualenv testApp
```

```
C:\kivy\testApp>python -m virtualenv testApp
created virtual environment CPython3.9.2.final.
0-64 in 3980ms
  creator CPython3Windows(dest=C:\kivy\testApp\
testApp, clear=False, no_vcs_ignore=False, glob
al=False)
  seeder FromAppData(download=False, pip=bundle
, setuptools=bundle, wheel=bundle, via=copy, ap
p_data_dir=C:\Users\dgposwojsku\AppData\Local\p
ypa\virtualenv)
    added seed packages: pip==21.0.1, setuptool
s==53.0.0, wheel==0.36.2
  activators BashActivator,BatchActivator,FishA
ctivator,PowerShellActivator,PythonActivator,Xo
nshActivator

C:\kivy\testApp>
```

Rys. 15.12. Tworzenie wirtualnego środowiska za pomocą Pythona

Buildozer

Dla osób używających jedynie Windowsa Buildozer będzie to większe wyzwanie niż dla użytkowników Linuksa, gdyż omawiana aplikacja ma jedynie wersję linuksową. Jeżeli na co dzień używasz systemu Windows, to masz kilka możliwości, by poradzić sobie z tym problemem:

1. Zainstaluj system Linux z Windows Store (dostępne m.in.: Ubuntu, Kali, SUSE, Debian, openSUSE Leap). Pamiętaj, że w ramach systemu operacyjnego Windows najpierw musisz uruchomić WSL2 (potrzebne informacje znajdziesz na stronie https://docs.microsoft.com/en-us/windows/wsl/install-win10), a dopiero potem (koniecznie po restarcie systemu) możesz zainstalować Linuksa. Zaleca się dystrybucję opartą na Debianie, gdyż ma on lepsze wsparcie zespołu Kivy, m.in. dokumentacja związana z omawianym procesem jest zdecydowanie lepiej opisana.
2. Użyj Linuksa w ramach wirtualizacji. Istnieje kilka rozwiązań tego typu, najpopularniejszym i bezpłatnym jest VirtualBox – szczegóły znajdziesz na stronie projektu https://www.virtualbox.org/. Pamiętaj, aby przed zainstalowaniem tego rozwiązania najpierw sprawdzić, czy w komputerze masz włączoną wirtualizację. Jeżeli nie, to musisz ją włączyć w systemowym BIOS.
3. Skorzystaj z Linuksa zainstalowanego na osobnym urządzeniu.

Najefektywniejszy jest oddzielny komputer z systemem Linux, gdyż proces buildowania jest obciążający czasowo i pod względem wykorzystywanych zasobów systemowych. Jeżeli korzystasz z oddzielnego komputera w trakcie opisywanych operacji, a ściślej gdy Buildozer to wykonuje, możesz wówczas równolegle pracować nad inną aplikacją.

Gdy już możesz korzystać z Linuksa, musisz wykonać kilka czynności:

1. Stworzyć katalog testAppAndr – w nim będziemy przeprowadzać wszystkie operacje (pamiętaj, aby go odblokować do czytania, pisania i zapisu).
2. Zainstalować narzędzie Buildozer (np. poleceniem **`pip install --user buildozer`**). Aby zadziałał, trzeba jeszcze doinstalować w systemie Linux co najmniej:
   - GIT

```
sudo apt-get install git
```

   - Cython

```
pip install Cython
```

   - kompilator Java (javac)

```
sudo apt-get install openjdk-8-jdk
```

   - SDK i NDK Androida – do pobrania ze strony https://developer.android.com.

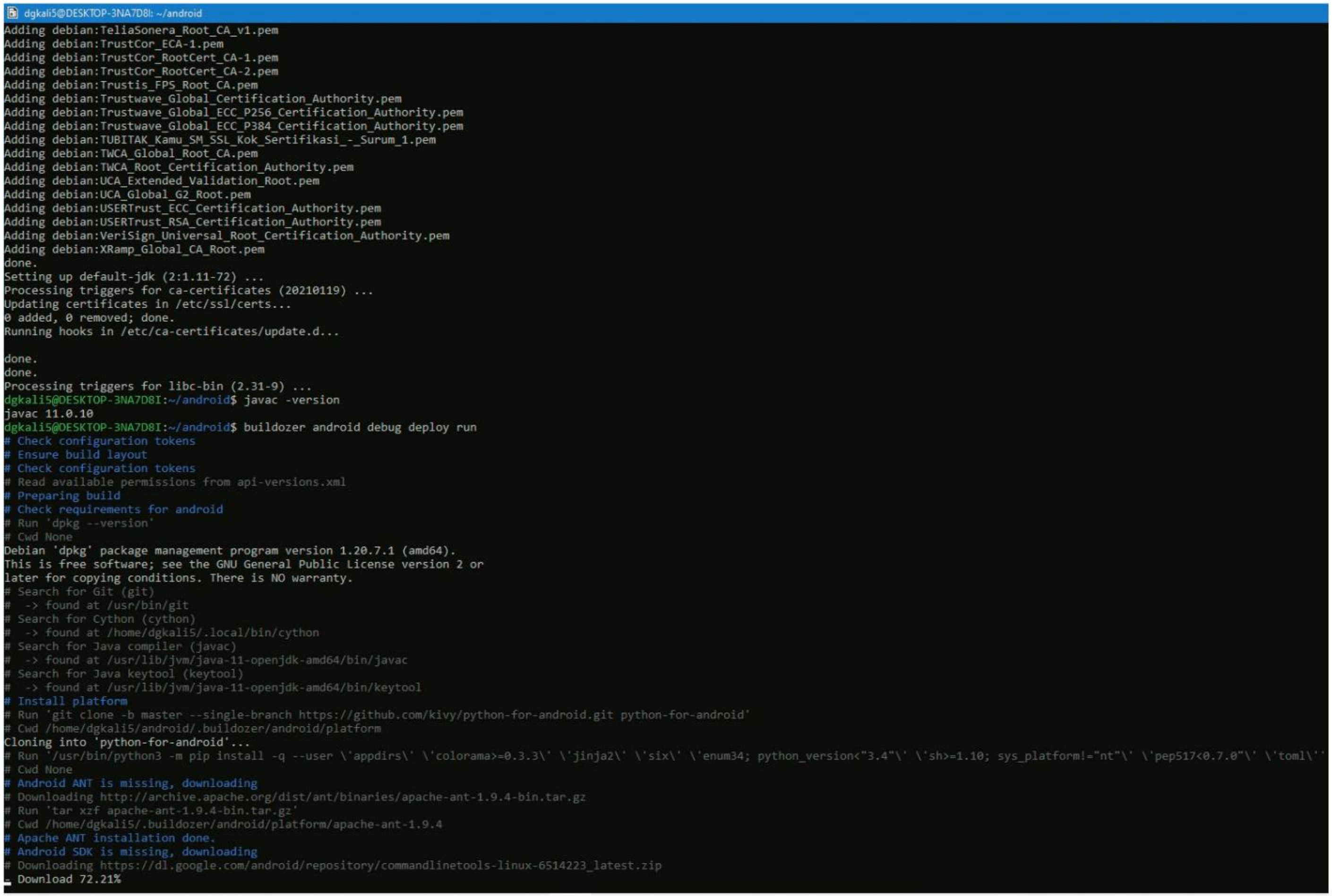
simpleKivy.py • simpleKivy.spec

```
# -*- mode: python ; coding: utf-8 -*-

block_cipher = None


a = Analysis(['simpleKivy.py'],
             pathex=['C:\\kivy\\testApp'],
             binaries=[],
             datas=[],
             hiddenimports=[],
             hookspath=[],
             runtime_hooks=[],
             excludes=[],
             win_no_prefer_redirects=False,
             win_private_assemblies=False,
             cipher=block_cipher,
             noarchive=False)
pyz = PYZ(a.pure, a.zipped_data,
             cipher=block_cipher)
exe = EXE(pyz,
          a.scripts,
          [],
          exclude_binaries=True,
          name='simpleKivy',
          debug=False,
          bootloader_ignore_signals=False,
          strip=False,
          upx=True,
          console=True )
coll = COLLECT(exe,
               a.binaries,
               a.zipfiles,
               a.datas,
               strip=False,
               upx=True,
               upx_exclude=[],
               name='simpleKivy')
```

Rys. 15.13. Widok wnętrz pliku .spec



Rys. 15.14. Uruchomienie narzędzia Buildozer – Debug and Deploy

Wymienione dodatkowe instalacje powinny uruchomić się samodzielnie w trakcie pracy z Buildozerem. Twoim zadaniem będzie wówczas głównie potwierdzanie chęci zainstalowania kolejnych narzędzi. Niestety, czasem pojawiają się tylko informacje o błędach – brakach, wówczas trzeba ręcznie doinstalować wymagane zależności.

Po skonfigurowaniu wszystkich zależności trzeba wykonać polecenie:

```
buildozer init

buildozer android debug deploy run
```

Warto wiedzieć, że detale wewnątrz instalacji Linuksa mogą spowodować różnego rodzaju komplikacje, które będą utrudniały ukończenie pracy aplikacji Buildozer. Gdy nie będziesz w stanie poradzić sobie z problemami, możesz się zwrócić o pomoc do społeczności internetowej i poszukać rozwiązania na specjalistycznych forach lub nawet zadać pytanie innym programistom.

Kivy Launcher

Aby z niego skorzystać i uruchomić projekt Kivy na urządzeniu z Androidem, musisz:

1. Zalogować się na urządzeniu mobilnym do sklepu Google Play i przejść do strony Kivy Launcher – https://play.google.com/store/apps/details?id=org.kivy.pygame.
2. Zainstalować aplikację KivyLauncher.
3. Nagrać stworzoną aplikację Kivy w katalogu na zewnętrznej karcie sdcard, włożonej do twojego smartfona, a następnie – po zainstalowaniu Kivy Launcher – uruchomić własne dzieło na urządzeniu mobilnym.
   Szczegóły znajdziesz na stronie: https://kivy.org/doc/stable/guide/packaging-android.html#packaging-your-application-for-the-kivy-launcher.

15.5. Alternatywne technologie tworzenia aplikacji mobilnych

Do tworzenia aplikacji mobilnych można wykorzystać różne technologie. Skupimy się na kilku wybranych. Zostaną one podzielone na dwie grupy:

- **technologie natywne**, czyli tworzone w językach programowania bezpośrednio związanych z danym systemem operacyjnym;
- **technologie hybrydowe**, czyli tworzone w różnorodnych środowiskach, które nie są bezpośrednio związane z daną technologią.

Technologie natywne

Na rynku są dostępne dwa podstawowe mobilne systemy operacyjne. W tabeli 15.1 przygotowano dla każdego z nich zestawy technologii do tworzenia aplikacji mobilnych.

Tabela 15.1. Zestawy technologii do tworzenia aplikacji mobilnych

Android	iOs
Strona projektu – https://www.android.com/	Strona projektu – https://www.apple.com/
System operacyjny z jądrem Linux dla urządzeń mobilnych, takich jak telefony komórkowe, smartfony, tablety	System operacyjny Apple Inc. dla urządzeń mobilnych iPhone bądź iPad
Technologia tworzenia aplikacji mobilnych to języki programowania Java i/lub Kotlin	Technologia tworzenia aplikacji mobilnych to języki programowania Object-C i/lub Swift
Środowisko IDE to program Android Studio – strona projektu: https://developer.android.com/studio	Środowisko IDE to program Xcode – strona projektu: https://developer.apple.com/xcode/

Technologie hybrydowe

Do tego rodzaju technologii należy m.in. framework Kivy. Warto wspomnieć o drugim frameworku Pythona – BeeWare (strona projektu: https://beeware.org/). Jest on nieco młodszy i może nie tak znany jak Kivy, ale są w nim wykonywane równie profesjonalne aplikacje mobilne. Na rys. 15.15 znajdziesz kody:

```
py -m venv beewareapp,

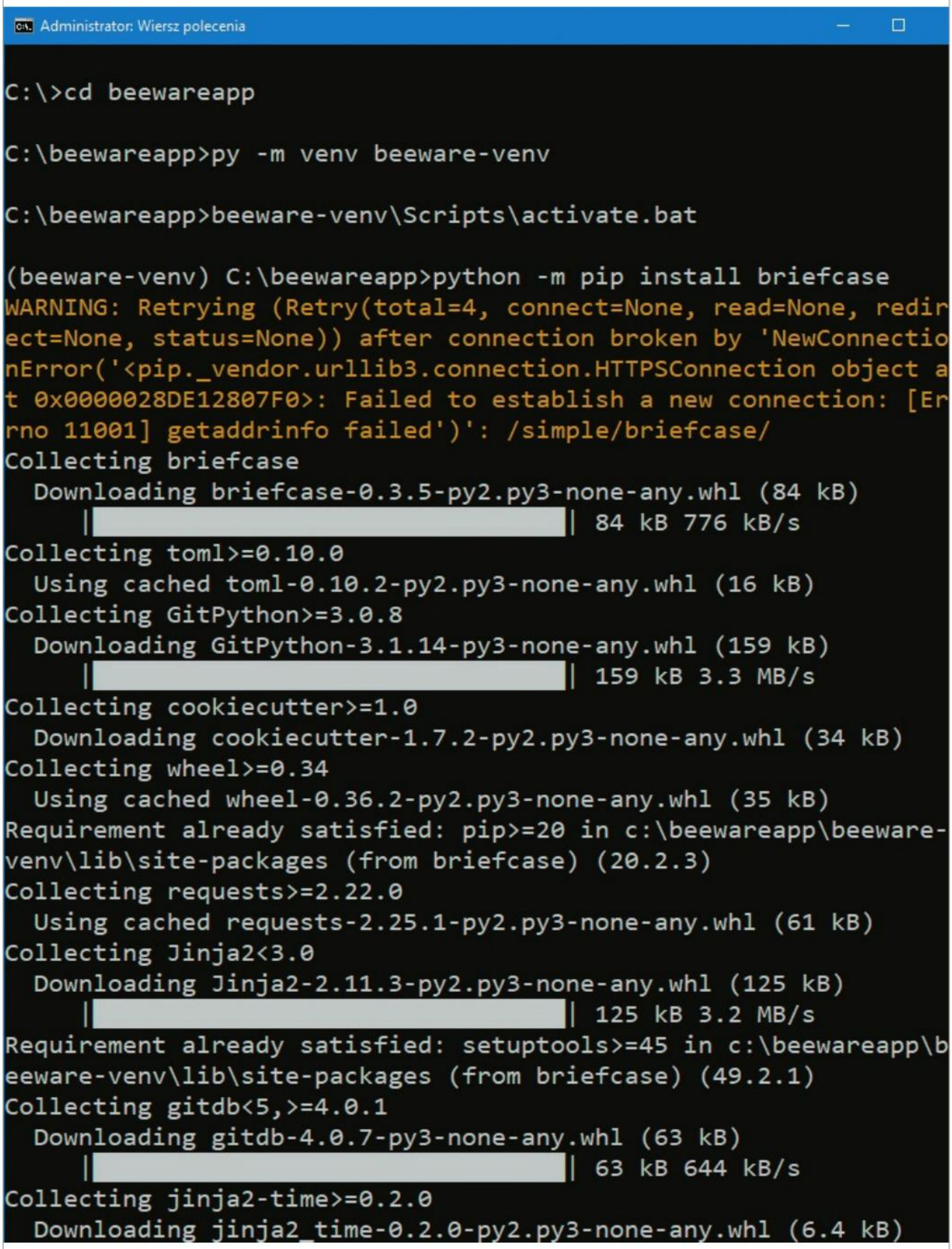
beeware-venv\Scripts\activate.bat,

python -m pip install briefcase,
```

które umożliwiają uruchomienie aplikacji opartej na BeeWare.

W tworzeniu aplikacji mobilnych dostępnych w innych językach niż Python królują różnego rodzaju frameworki, dzięki którym można w stosunkowo szybki sposób przygotować aplikację mobilną. Zaletą takiego rozwiązania jest to, że do obsługi zarówno systemu Android, jak i iOS potrzebny będzie – zwykle w 80–90% – jeden kod. Reszta kodu, ze względu na specyfikę poszczególnych systemów operacyjnych urządzeń mobilnych, musi być jednak nieco inna na każdą z wymienionych platform. Ma to swoje olbrzymie zalety, gdyż podczas programowania aplikacji od razu powstaje uniwersalna platforma, która po właściwym uzupełnieniu umożliwi działanie dzieła na wymienionych systemach operacyjnych.

Aplikacje hybrydowe są zwykle nieco wolniejsze niż te wcześniej wymienione – natywne. Nie wszystkie opcje związane z dostępem do sprzętu są osiągalne z poziomu aplikacji hybrydowych. Znacznie różnią się one od siebie w poszczególnych technologiach. Warto wymienić najważniejszych przedstawicieli tego sposobu tworzenia aplikacji mobilnych.



Rys. 15.15. Przygotowanie do uruchomienia przykładowej aplikacji frameworka BeeWare

JavaScript:

- React Native, strona domowa – https://reactnative.dev/,
- NativeScript, strona domowa – https://nativescript.org/,
- Apache Cordova, strona domowa – https://cordova.apache.org/,
- IONIC, strona domowa – https://ionicframework.com/,
- Mobile Angular UI, strona domowa – http://mobileangularui.com/,
- Sencha Touch, strona domowa – https://www.sencha.com/products/touch/,
- Kendo UI, strona domowa – https://www.telerik.com/kendo-ui,
- Framework7, strona domowa – https://framework7.io/.

```
import React, {Fragment} from 'react';
import {
  SafeAreaView,
  StyleSheet,
  ScrollView,
  View,
  Text,
  StatusBar,
} from 'react-native';
```

Rys. 15.16. Przygotowanie do uruchomienia frameworka React Native

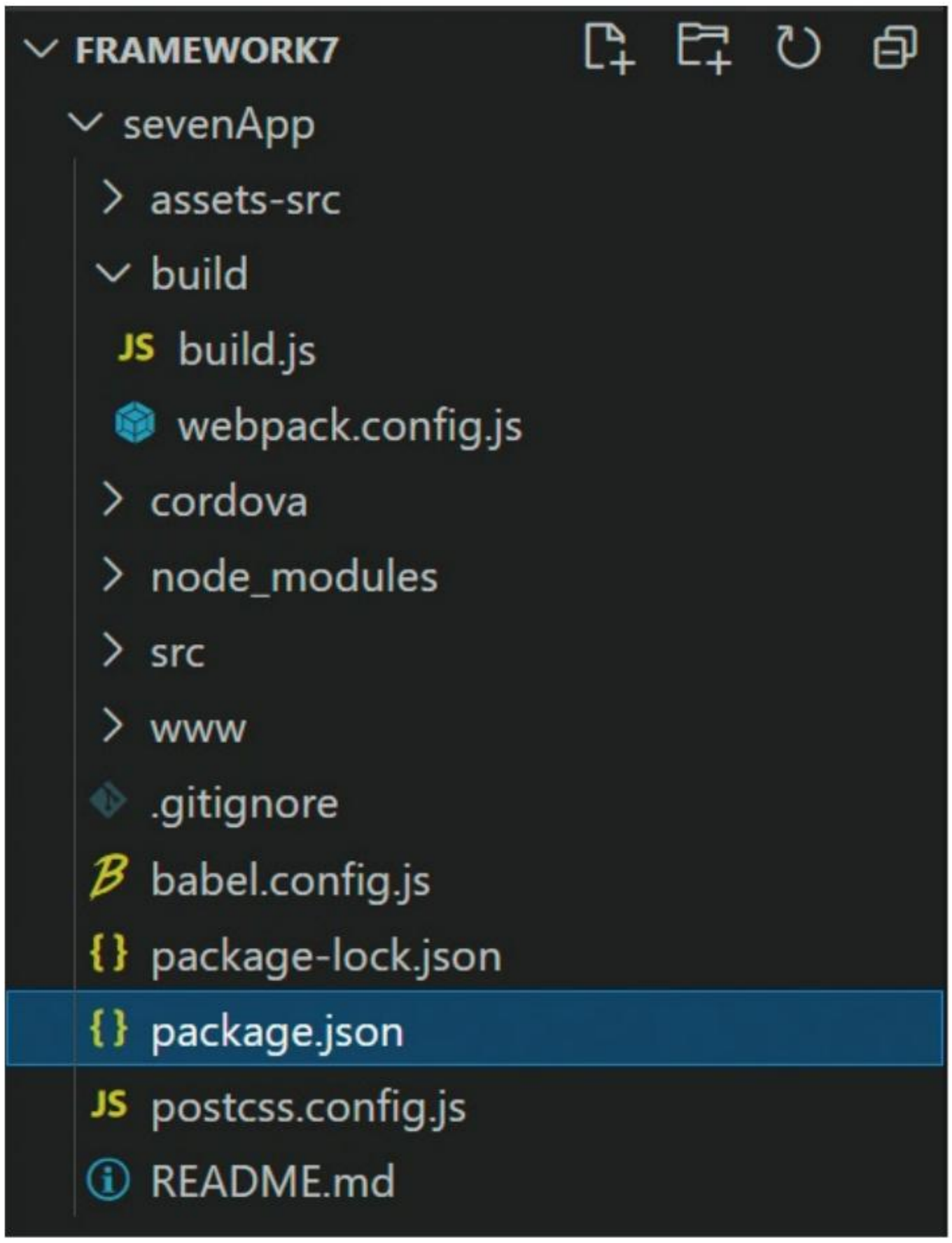
CIEKAWOSTKA

Framework7 to jedno z ciekawszych i bezpłatnych rozwiązań pozwalających na budowanie aplikacji mobilnej z elementów przygotowanych w ramach frameworka i gotowych do użycia (strona domowa projektu: https://framework7.io/). Na rys. 15.17 przedstawiono widok plików podstawowych po zainstalowaniu aplikacji.

Zaznaczony na niebiesko plik package.json jest jednym z kluczowych elementów w budowie zestawu niezbędnych modułów i zależności tego frameworka. Instalacja omawianej aplikacji jest bardzo prosta i składa się zaledwie z dwóch poleceń, które trzeba wykonać w dowolnym terminalu bądź konsoli:

- npm i framework7-cli cordova –g,
- framework7 create --ui.

Aplikacja framework7 jest oparta na języku programowania JavaScript i udostępnia bardzo bogaty ekosystem, współpracujący z wieloma innymi aplikacjami, m.in. Vue, React, Electron, NW.js, Svelte, ApacheCordova.



Rys. 15.17. Framework7 – widok katalogów i plików projektu aplikacji mobilnej

Niestety, nie wszystkie wymienione aplikacje są bezpłatne. Pierwsze trzy spośród wskazanych są jednymi z najpopularniejszych i dają największe możliwości dostępu do poszczególnych elementów urządzeń mobilnych. Dodatkowym ich atutem jest brak konieczności uiszczania opłat związanych z ich komercyjnym użytkowaniem.

C# – platforma Xamarin

Do programowania aplikacji mobilnych można także użyć frameworka Xamarin (strona domowa: https://dotnet.microsoft.com/apps/xamarin).

Aby tworzyć w nim aplikacje, musisz poznać:

- język znaczników XAML (więcej informacji na https://docs.microsoft.com/en-us/dotnet/desktop/wpf/fundamentals/xaml);
- język programowania C# (więcej informacji na https://docs.microsoft.com/pl-pl/dotnet/csharp/); przykład kodu przedstawiono na rys. 15.19.

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/
forms"
             xmlns:x="http://schemas.microsoft.com/
             winfx/2009/xaml"
             x:Class="Notes.NoteEntryPage"
             Title="Note Entry">
    <StackLayout Margin="20">
        <Editor Placeholder="Wprowadź notatkę"
                Text="{Binding Text}"
                HeightRequest="100" />
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <Button Text="Save"
                    Clicked="OnSaveButtonClicked" />
            <Button Grid.Column="1"
                    Text="Delete"
                    Clicked="OnDeleteButtonClicked"/>
        </Grid>
    </StackLayout>
</ContentPage>
```

Rys. 15.18. XAML – przykład kodu elementu aplikacji mobilnej

Aplikacja Xamarin opiera się na technologii .NET (szczegóły znajdziesz na stronie domowej projektu: https://dotnet.microsoft.com/). Jest to platforma deweloperska składająca się z wielu narzędzi, języków programowania i bibliotek do tworzenia różnych typów i rodzajów aplikacji. Całość wskazanego pakietu dysponuje narzędziami i bibliotekami umożliwiającymi tworzenie aplikacji dla systemów Android, iOS, tvOS, watchOS, macOS i Windows.

```
using System;
using System.IO;
using Xamarin.Forms;

namespace Notes
{
    public partial class App : Application
    {
        public static string FolderPath { get;
        private set; }

        public App()
        {
            InitializeComponent();
            FolderPath = Path.Combine(Environment.
            GetFolderPath(Environment.SpecialFolder.
            LocalApplicationData));
            MainPage = new NavigationPage(new
            NotesPage());
        }
        // ...
    }
}
```

Rys. 15.19. Przykład kodu elementu aplikacji mobilnej – język C#, framework Xamarin

Niewątpliwym ułatwieniem w tworzeniu aplikacji mobilnych z użyciem Xamarin, XAML i C# jest możliwość wykorzystania aplikacji RAD, czyli Visual Studio, np. Community Edition 2019 (strona domowa aplikacji: https://visualstudio.microsoft.com/pl/).

Dart

Jest to język specjalizujący się w potrzebach tworzenia interfejsu użytkownika (strona domowa projektu: https://dart.dev/) wraz z frameworkiem Flutter (strona domowa projektu: https://flutter.dev/).

Powyższy zestaw to jedno z najmłodszych rozwiązań na rynku technologii mobilnych, który dość szybko zyskuje na popularności, na co zapewne (poza jakością rozwiązania) ma wpływ to, że zostało wprowadzone przez firmę Google (https://www.google.com/) i jest przez nią mocno wspierane. Stworzyła się wokół niego prężna społeczność programistów, którzy wspomagają swoimi pomysłami (i pracą) rozwój tego projektu.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co to jest framework Kivy? Do czego służy?
2. Wyjaśnij pojęcie aplikacji mobilnej.
3. Opisz proces przygotowania pakietów, które mogą być uruchomione w systemie Android.
4. Wymień co najmniej cztery systemy operacyjne, na które można przygotować pakiety za pomocą frameworka Kivy.
5. Przedstaw co najmniej trzy technologie tworzenia aplikacji mobilnych alternatywne dla frameworka Kivy.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Wykonaj polecenia:
   A. Zainstaluj w komputerze środowisko Python 3.6+.
   B. Stwórz katalog testPy, a wewnątrz niego zainstaluj środowisko wirtualne języka programowania Python.
   C. Wewnątrz katalogu testPy stwórz plik testPy.py. Napisz w nim program, który wyświetli w konsoli napis: „Witaj w środowisku wirtualnym Pythona”.
2. Wykonaj polecenia.

Dla komputerów z systemem operacyjnym Windows

Zainstaluj w komputerze system operacyjny Linux Ubuntu:

a) wewnątrz VirtualBox (pamiętaj o wcześniejszym sprawdzeniu, czy wirtualizacja na komputerze nie jest zablokowana i jest dostępna dla danego procesora) lub
b) skorzystaj z instalacji ze sklepu Microsoft Store (pamiętaj o wcześniejszym zainstalowaniu technologii WSL2).

Dla komputerów z systemem operacyjnym Linux oraz Windows

a) W systemie operacyjnym Linux Ubuntu zainstaluj aplikację Buildozer.
b) Za pomocą programu Buildozer przygotuj wersję mobilną na system operacyjny Android programu zapisanego w pliku zadania 1. – testPy.py.

Jeżeli nie udało się przeprowadzić procesu buildowania, przygotuj raport opisujący przyczyny niepowodzenia.

16 Programowanie aplikacji mobilnych opartych na języku programowania Python

ZAGADNIENIA

- Kivy – wprowadzenie do frameworka
- UI, GUI, WUI – wyjaśnienie podstawowych pojęć
- Elementy UI aplikacji mobilnej
- Formularz – projekt aplikacji opartej na GridLayout i Python
- Język kv – projektowanie graficzne aplikacji
- Projekt aplikacji – Stoper
- Współpraca Kivy z relacyjną bazą danych SQL
- Ekrany aplikacji mobilnej
- Przygotowanie aplikacji do publikacji w sklepie

16.1. Kivy – wprowadzenie do frameworka

Aby zainstalować framework, w konsoli trzeba wykonać polecenie:

```
python -m pip install kivy==2.0.0
```

Decyzja dotycząca instalowanej wersji należy do każdego programisty.

Za pomocą polecenia konsolowego:

```
pip freeze
```

zobaczysz zależności, czyli programy wraz z numerami wersji, które masz na swoim komputerze. Na rys. 16.1 przedstawiono zależności, które znajdują się na przykładowym komputerze w trakcie przygotowywania materiałów.

UWAGA

W przypadku instalacji frameworka Kivy poleceniem:

```
python -m pip install kivy[full] kivy_examples
```

- gdy znajdujesz się w danej lokalizacji, np. wewnątrz katalogu `kivy`, będącym bezpośrednio na dysku C, to framework zostanie zainstalowany wewnątrz znajdującej się Windowsie instalki Pythona – konkretny katalog zależy od wersji systemu operacyjnego i zainstalowanego Pythona;
- gdy znajdujesz się w danej lokalizacji, np. wewnątrz katalogu `kivy`, będącym bezpośrednio na dysku C, i uruchomisz tam środowisko wirtualne Pythona oraz – będąc w tym środowisku – wykonasz instalację, to framework zostanie zainstalowany wewnątrz danego katalogu, czyli w omawianym tutaj przykładzie wewnątrz folderu `kivy`.

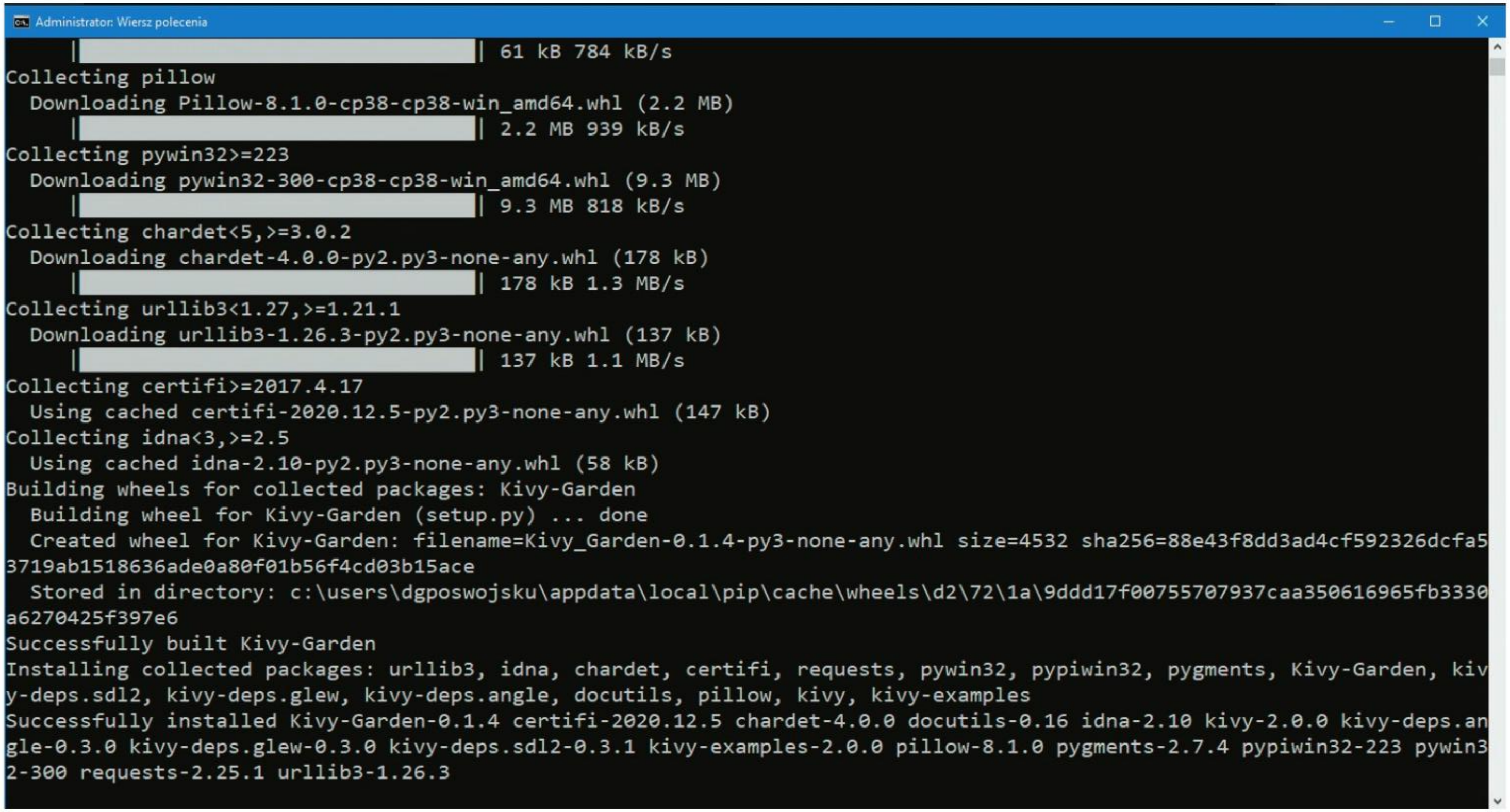
```
PS C:\kivy> pip freeze
altgraph==0.17
certifi==2020.12.5
chardet==4.0.0
docutils==0.16
future==0.18.2
idna==2.10
Kivy==2.0.0
kivy-deps.angle==0.3.0
kivy-deps.glew==0.3.0
kivy-deps.gstreamer==0.3.1
kivy-deps.sdl2==0.3.1
Kivy-examples==2.0.0
Kivy-Garden==0.1.4
pefile==2019.4.18
Pillow==8.1.0
Pygments==2.8.0
pyinstaller==4.2
pyinstaller-hooks-contrib==2020.11
pypiwin32==223
pywin32==300
pywin32-ctypes==0.2.0
requests==2.25.1
urllib3==1.26.3
PS C:\kivy>
```

Rys. 16.1. Efekt działania kodu **`pip freeze`**

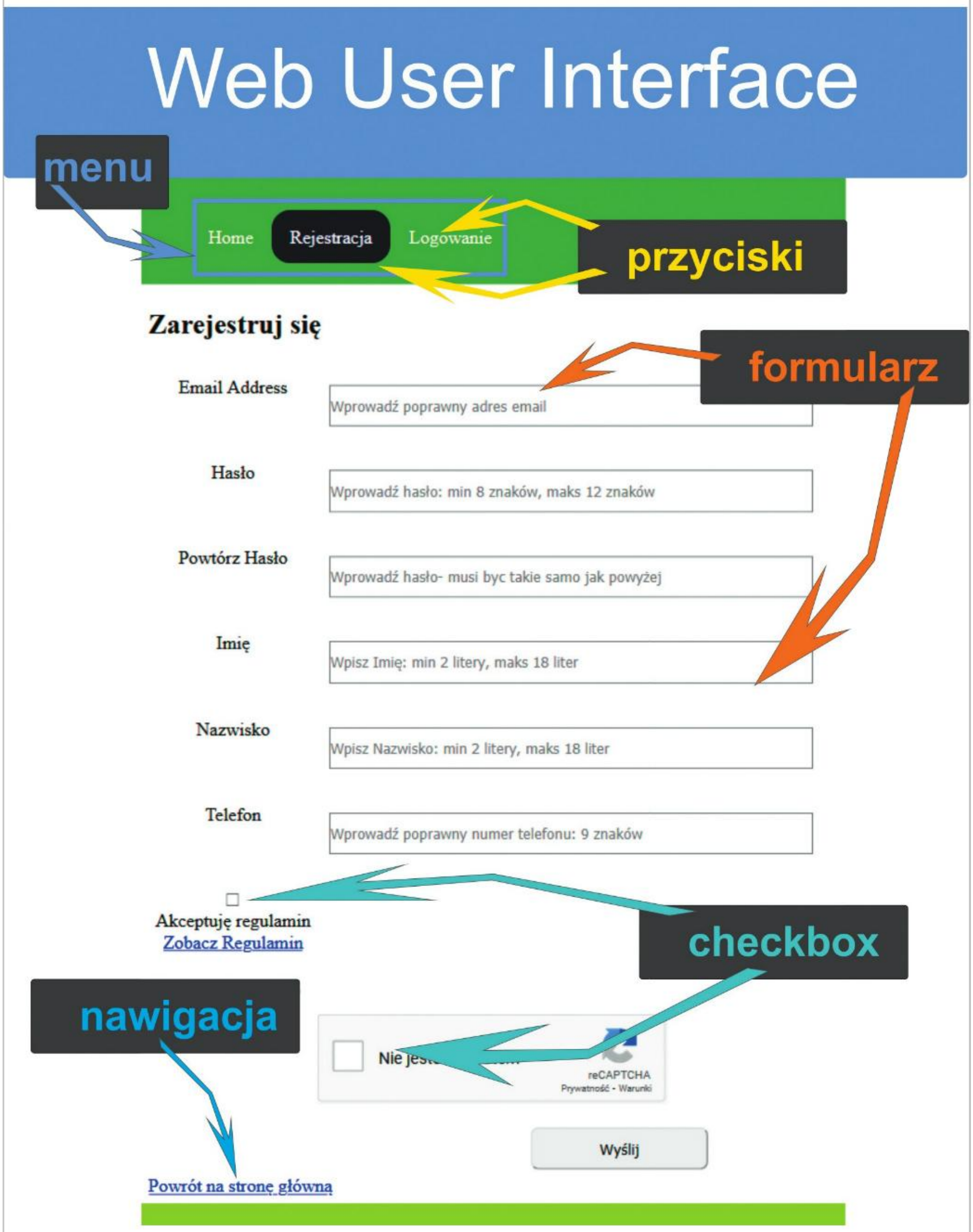
Na początek przygotujemy prostą aplikację, szkielet, do którego będzie można dodać elementy GUI.

16.2. UI, GUI, WUI – wyjaśnienie podstawowych pojęć

UI (ang. *user interface*) – **interfejs użytkownika** to część oprogramowania, która jest odpowiedzialna za interakcję z użytkownikiem, a ściślej za połączenie poziomu i sposobu postrzegania oraz użytkowania rzeczywistości przez człowieka i maszynę. Istnieje kilka możliwych „miejsc styku" człowieka i maszyny. Może się to odbywać w różnych środowiskach, ale zawsze chodzi o to samo: aby człowiek miał dostępne takie narzędzia, które będzie w stanie zrozumieć i skutecznie używać. Najlepiej, aby były one maksymalnie intuicyjne w użytkowaniu.



Rys. 16.2. Konsola CMD – najprostszy przykład interfejsu użytkownika (w połączeniu z myszką / klawiaturą i ekranem)



Rys. 16.3. Przykład kilku elementów graficznego interfejsu użytkownika (GUI) – tutaj na stronie internetowej (WUI)

Warto się przyjrzeć kilku obszarom „styku" człowieka i komputera:

- Konsola (terminal) – wiersz poleceń (CLI, ang. *Command Line Interface*), w programowaniu niesamowicie przydatne, zwłaszcza gdy doda się do tego:
  - klawiaturę i myszkę jako urządzenia wejściowe;
  - monitor (wyświetlacz) w trybie znakowym lub drukarka jako urządzenia wyjściowe.
- Interfejs graficzny (GUI) – wejście, tutaj jest nim zwykle myszka i/lub klawiatura, a wyjściem (kolorowy) wyświetlacz graficzny, czyli np. monitor komputerowy bądź ekran urządzenia przenośnego.
- Interfejs projektu internetowego (WUI, ang. *Web User Interface*) – portalu, strony, aplikacji webowej. W tym przypadku wejściem i wyjściem są elementy projektu webowego wyświetlane w wybranej przeglądarce.
- Interfejs głosowy (VUI, ang. *Voice User Interface*) – dzięki technologii rozpoznawania mowy jest możliwy do zastosowania prawie w każdej dziedzinie życia. Urządzeniem wejściowym jest mikrofon, a urządzeniem wyjściowym – głośniki.
- Interfejs gestowy / ruchowy – jedno z najnowszych rozwiązań. Urządzenie wejściowe to ciało ludzkie bądź specjalny kontroler w połączeniu z czujnikami ruchu. Wyjściem jest w tym przypadku wyświetlacz graficzny – przykładem mogą być konsole do gier.

WUI to właściwie jeden z rodzajów GUI, dlatego rzadko można spotkać się z tym określeniem. Zdecydowana większość programistów używa skrótowca GUI, także w odniesieniu do świata aplikacji i stron internetowych. GUI to rozwinięcie pojęcia UI przez dodanie do niego elementu graficznego.

16.3. Elementy UI aplikacji mobilnej

Przykładowe elementy (interfejs użytkownika), z którymi można mieć do czynienia w nowoczesnej aplikacji mobilnej:

- przyciski,
- nawigacja,
- okna dialogowe,
- listy,
- formularze,
- paski narzędziowe,
- grafika,
- animacje,
- dźwięk.

Popularne nowoczesne frameworki do wykonywania aplikacji mobilnych zwykle dostarczają już gotowych elementów. Zauważ, że mamy do czynienia z hermetyzacją. Nie zaglądamy do środka klas zawartych we frameworku Kivy, tylko korzystamy z jego funkcjonalności i wyglądu. Jest to wygodne i praktyczne, choć zmusza do trzymania się zasad twórców danego frameworka. Tak samo jest w przypadku Kivy. W zamian za mniejszą elastyczność otrzymujemy elementy już gotowe do użycia. Przykładowo: żeby uzyskać zwykły przycisk, wystarczy kilka linijek kodu (.kv + .py) zamiast kilkudziesięciu (tak byłoby, gdybyśmy wszystko tworzyli wyłącznie kodem Pythona).

```
BoxLayout:
    padding: '3sp'
    canvas:
        Color:
            rgba: 0, 1, 1, .6
        Rectangle:
            size: self.size
            pos: self.pos
    size_hint: 1, None
    height: '25sp'
    Spinner:
        size_hint: None, 1
        width: '108sp'
        text: 'Witaj'
    Widget:
    BoxLayout:
        size_hint: None, 1
        width: '40sp'
        Label:
            text: "Witaj w języku kv"
        CheckBox:
            id: chkbx
            active: True
            size_hint_x: 1
```

Rys. 16.4. Przykład kodu języka kv, który jest własnym językiem programowania frameworka

Tworząc aplikację we frameworku Kivy, trzeba na początku rozważyć co najmniej dwa zagadnienia.

ZAGADNIENIE 1.

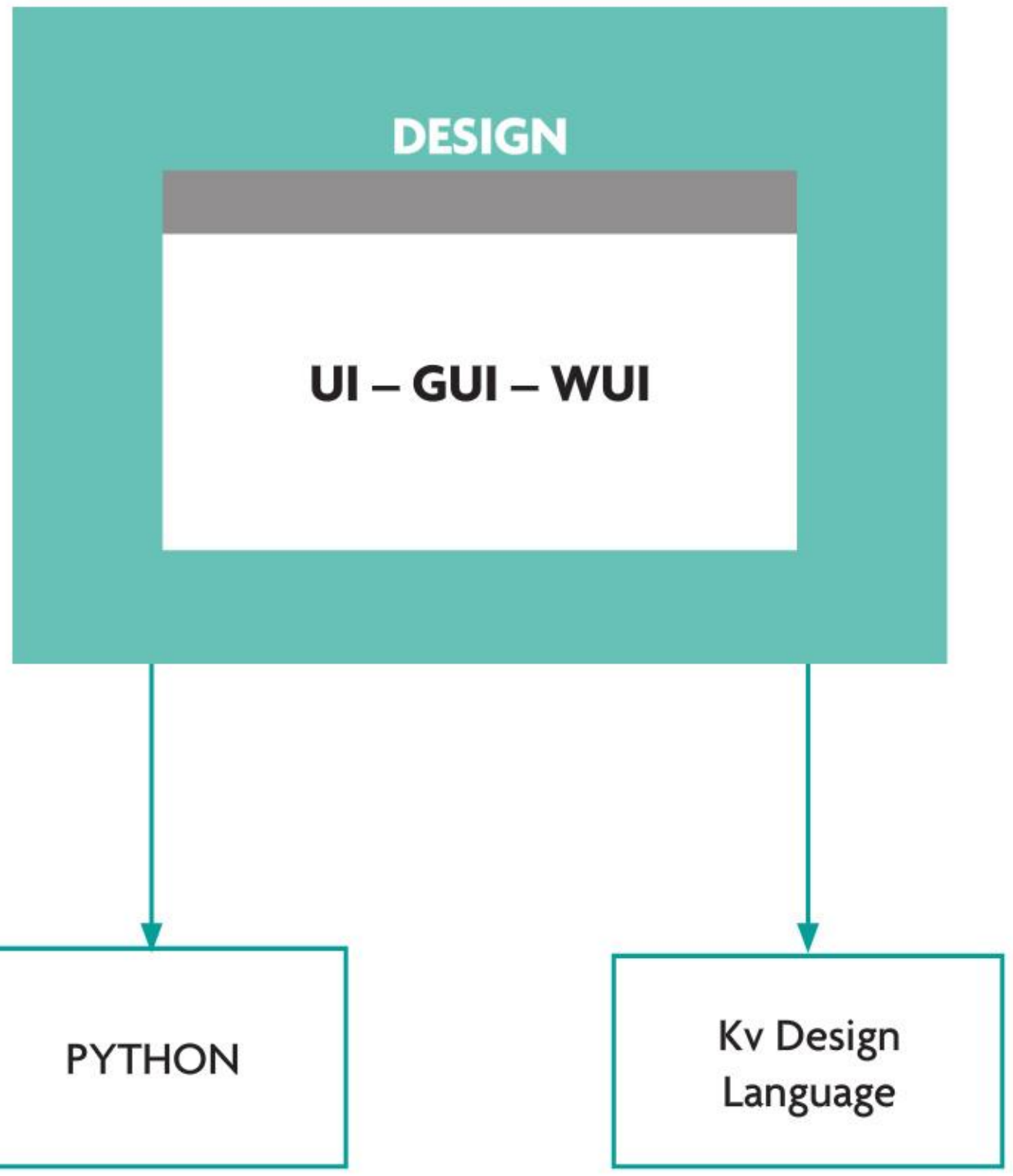
Jaka funkcjonalność będzie nam potrzebna.

ZAGADNIENIE 2.

W jaki sposób będziemy tworzyli elementy układu strony, czyli UI, używając:

- Pythona;
- języka kv; jest to wewnętrzny język frameworka Kivy.

Na stronie projektu **Kv Design Language** (języka projektowania kv) można przeczytać: „Kivy zapewnia język projektowania specjalnie nastawiony na łatwy i skalowalny projekt GUI. Język kv ułatwia oddzielenie projektu interfejsu od logiki aplikacji" (źródło: https://kivy.org/doc/stable/gettingstarted/rules.html)".



Rys. 16.5. Dwie metody wykonania wyglądu aplikacji tworzonej we frameworku Kivy

Funkcjonalność frameworka Kivy bierze się z:

- języka Python;
- wnętrza frameworka, czyli modułów Kivy, które dają nam określoną funkcjonalność, np. potrafią stworzyć etykietę (Label).

Na rys. 16.6 przedstawiono import przykładowych modułów Kivy.

```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
from kivy.graphics import Color, Rectangle
from kivy.uix.bubble import Bubble
from kivy.uix.switch import Switch
from kivy.uix.popup import Popup
from kivy.uix.image import Image
from kivy.uix.scatter import Scatter
from kivy.core.image import Image as CoreImage
from kivy.lang import Builder
```

Rys. 16.6. Kod z pliku frameworka Kivy – import przykładowych modułów zawierających określoną funkcjonalność

Moduły Kivy, które udostępniają programiście określoną funkcjonalność, trzeba zaimportować do projektu na początku pliku. Aby w programie móc użyć odniesienia do wspomnianej etykiety:

```
self.add_widget(Label(text="To jest ETYKIETA ")),
```

najpierw trzeba zaimportować funkcjonalność etykiety (Label):

```
from kivy.uix.label import Label
```

Aby wstawić pole Input, czyli możliwość wpisania np. loginu przez użytkownika:

```
self.name = TextInput(multiline=False)
self.add_widget(self.name),
```

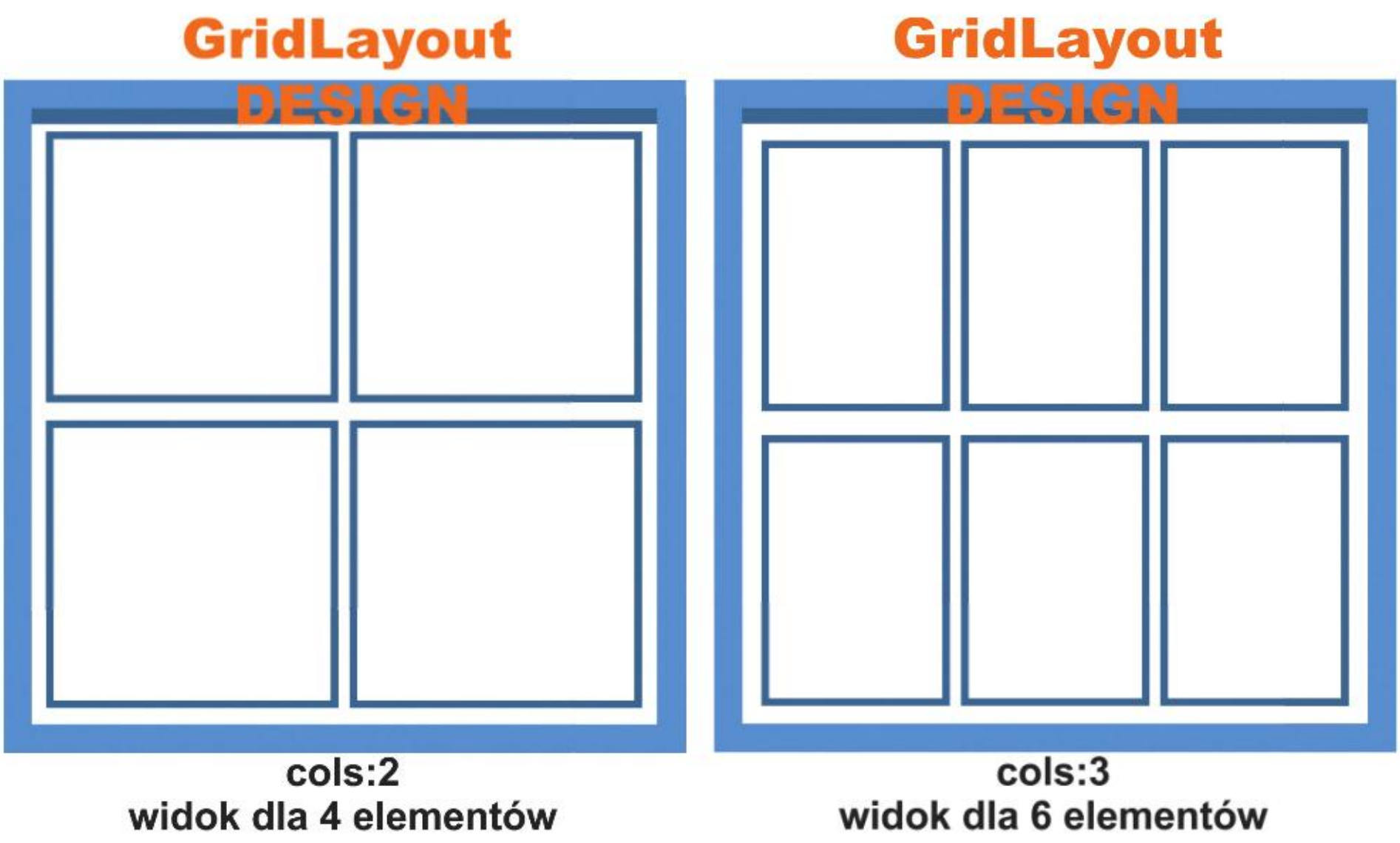
najpierw trzeba zaimportować odpowiednią funkcjonalność:

```
from kivy.uix.textinput import TextInput
```

Layout – układy strony

Layouty to kontenery używane do rozmieszczenia wewnątrz nich – w określony sposób – widżetów. Istnieje kilka przygotowanych układów strony, które można użyć po ich zaimportowaniu. Oto najczęściej stosowane opisy:

- **AnchorLayout**
  - widżety można zakotwiczyć na górze, na dole, po lewej, po prawej lub na środku;
- **BoxLayout**
  - widżety są rozmieszczone sekwencyjnie, w orientacji pionowej lub poziomej;
- **RelativeLayout**
  - widżety podrzędne są umieszczane względem układu;
- **GridLayout**
  - widżety są rozmieszczone w siatce zdefiniowanej przez:
    - wiersze – właściwość rows,
    - kolumny – właściwość cols;
- **PageLayout**
  - służy do tworzenia prostych układów wielostronicowych w sposób umożliwiający łatwe przechodzenie z jednej strony na drugą za pomocą obramowań.



Rys. 16.7. Układ strony GridLayout

Poza układami strony w UI Kivy[1] jest dostępnych wiele elementów różnego rodzaju – od prostych przycisków i etykiet po elementy zaawansowane, takie jak accordion, slider, zegar.

[1] Oficjalna dokumentacja Kivy – *Kivy Documentation Release 2.1.0.dev0* zawiera 755 stron tekstu z przykładami. Możesz ją pobrać ze strony https://kivy.org/doc/stable/.

16.4. Formularz – projekt aplikacji opartej na GridLayout i Pythonie

Zaprezentujemy tu:

- zastosowanie importu odpowiednich modułów;
- użycie układu strony GridLayout;
- umieszczenie na stronie kilku podstawowych elementów jako widgetów;
- dodanie interakcji – po wpisaniu tekstu w elemencie UI Input zostanie on przechwycony do innego obszaru aplikacji.

Zaczniemy od importu niezbędnych modułów Kivy, obsługujących pożądane w projekcie funkcjonalności.

```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
from kivy.uix.image import Image
```

Rys. 16.8. Układ strony GridLayout zawierający: Label, Input, Image, Button – import modułów

Gdy moduły już udostępniają w naszym projekcie elementy funkcjonalności Kivy, trzeba zaprogramować klasę Formularz, do której jako argument jest przekazywany układ strony GridLayout. Po inicjalizacji klasy będzie można dodać kolejne elementy na stronie. W naszym przykładzie zrobimy to za pomocą kodu Pythona, który doda odpowiednie części UI, jako widgety, do wnętrza układu GridLayout. Ponieważ ustawimy parametr self.cols na 2, otrzymamy dwie kolumny. Każdy kolejny widget będzie dodawany po poprzednim, czyli:

- nieparzyste (od pierwszego) – będą w lewej kolumnie;
- parzyste (od drugiego) – będą w prawej kolumnie.

Jest to bardzo praktyczne rozwiązanie, które samoczynnie układa elementy na stronie. W naszym przypadku kolumny będą takiej samej szerokości. Oczywiście można użyć stosownych parametrów, aby zrobić jedną z nich szerszą. Jednym ze sposobów może być zmiana rozmiarów elementów:

- nieparzystych na takie same, np.:
  - elementowi UI – Button nadać właściwość **`'width: root.width*0.2'`**;
- parzystych na takie same, np.:
  - elementowi UI – Button nadać właściwość **`'width: root.width*0.67'`**.

Powyższe ustawienia spowodowałyby, że lewa kolumna aplikacji będzie zajmowała 20% dostępnej szerokości, a prawa – 67% szerokości.

```
# Stworzenie klasy apki Formularz, której układ strony
wykorzystuje GridLayout
class Formularz(GridLayout):

    # Inicjalizacja apki
    def __init__(self, **kwargs):
        # Call grid layout konstruktor
        super(Formularz, self).__init__(**kwargs)

        # Wstaw kolumny
        self.cols = 2
```

Rys. 16.9. Układ strony GridLayout zawierający: Label, Input, Image, Button – budowa klasy Formularz

Po zdefiniowaniu liczby kolumn tworzymy za pomocą języka Python i frameworka Kivy kolejne elementy UI na stronie, a z nich widgety.

Powstaną:

- Label – etykieta, wpisanie tekstu;
- TextInput – pole do wpisywania przez użytkownika znaków z klawiatury;
- Image – element UI pozwalający na dodanie obrazka;
- Button – przycisk pozwalający m.in. na wywołanie określonej funkcji (jeszcze nie jest zaprogramowany na dane zdarzenie).

W ramach stworzonego kodu dodano także pewne delikatne elementy upiększające widgety:

- Label – otrzymał właściwość **`font_size = "3psp"`**, czyli nadaną wielkość czcionki;
- TextInpu – otrzymał właściwość **`background_color = [1, 1, 0, 1]`**, kolor;
- Button – otrzymał właściwość **`background_color = [1, 0, 0, 1]`**, kolor (rys. 16.10).

Na koniec stworzymy jedną klasę o nazwie MobileApp, która zwraca **`Formularz()`**. Całość jest uruchamiana konstrukcją:

```
if __name__ == '__main__':
    MobileApp().run()
```

Zwróć uwagę, że na górnej etykiecie uruchomionej aplikacji pojawił się napis Mobile. Zostaje on zaczerpnięty z pierwszej części nazwy klasy **`MobileApp()`**.

```
        # Dodaj widget z Label (etykietą)
        self.label = self.add_widget(Label(text="To
        jest ETYKIETA ",font_size ="30sp"))

        # Dodaj Pole Input (Box)
        self.name = TextInput(multiline=False)
        self.name.background_color = [1, 1, 0, 1]
        # Dodaj widget z obiektem - polem input
        self.add_widget(self.name)

        # Dodaj Image
        self.img = Image(source="rys.png")
        # Dodaj widget z obiektem obrazkowym
        self.add_widget(self.img)
```

Rys. 16.10. Układ strony GridLayout zawierający: Label, Input, Image, Button – dodanie widgetów

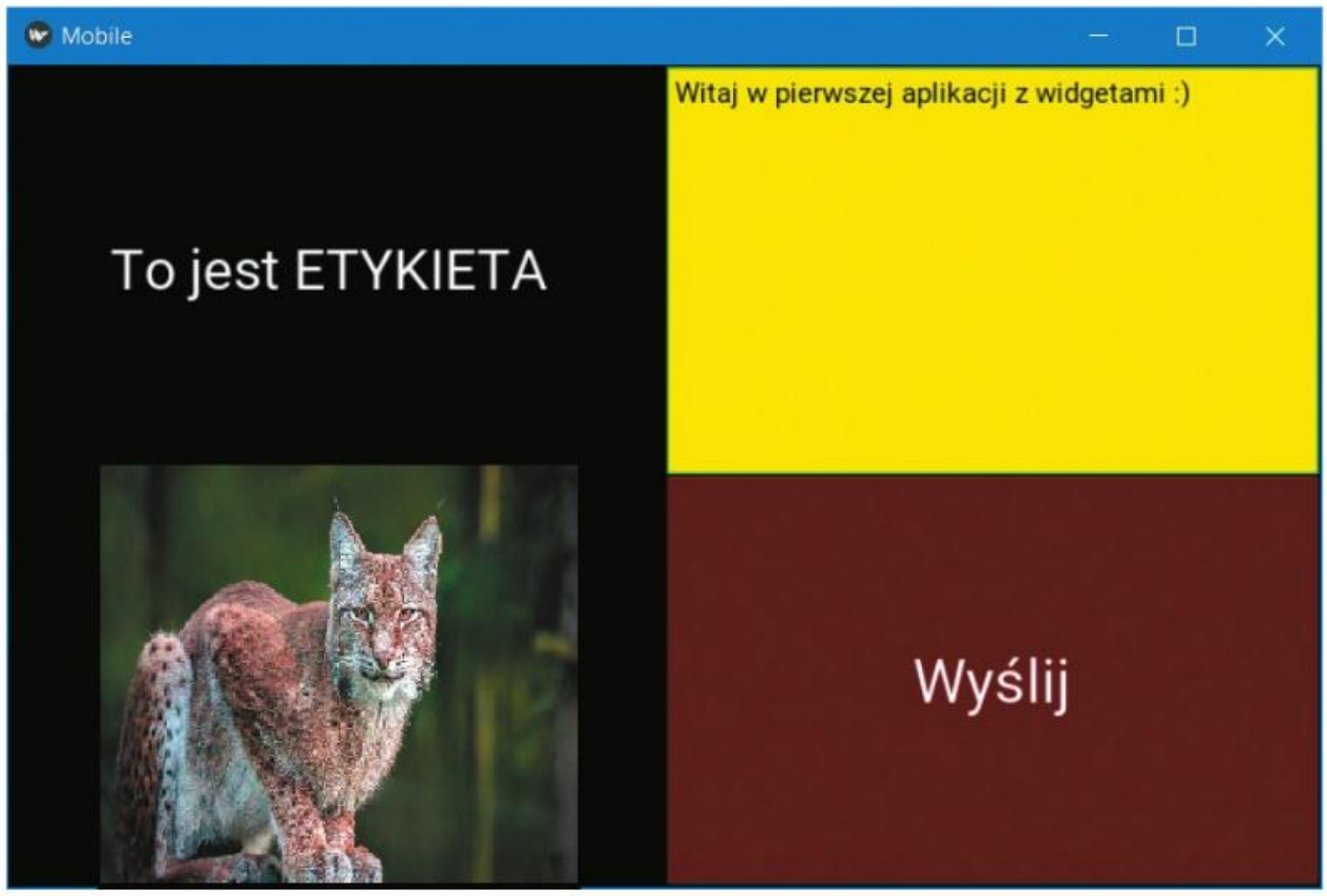
```
        # Przycisk do wysyłania (Submit
        Button)
        self.submit = Button(text="Wyślij",
        font_size=32)
        self.submit.background_color = [1, 0,
        0, 1]
        # Dodaj widget - przycisk do
        wysyłania (Submit Button)
        self.add_widget(self.submit)

class MobileApp(App):
    def build(self):
        return Formularz()

if __name__ == '__main__':
    MobileApp().run()
```

Rys. 16.11. Układ strony GridLayout zawierający: Label, Input, Image, Button – uruchomienie aplikacji

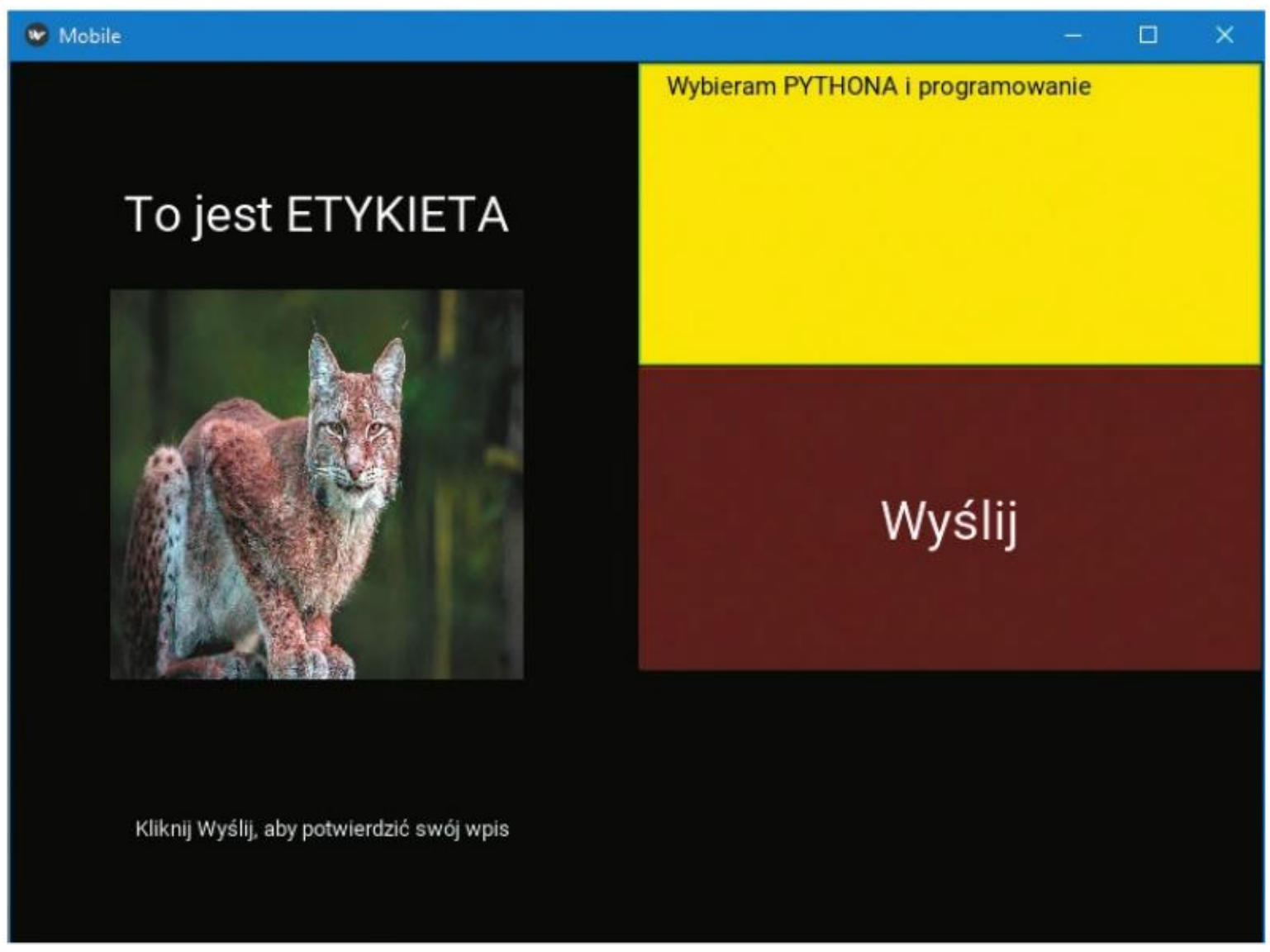
Na rys. 16.12 przedstawiono wygląd uruchomionej aplikacji mobilnej.



Rys. 16.12. Układ strony GridLayout zawierający: Label, Input, Image, Button – widok uruchomionej aplikacji

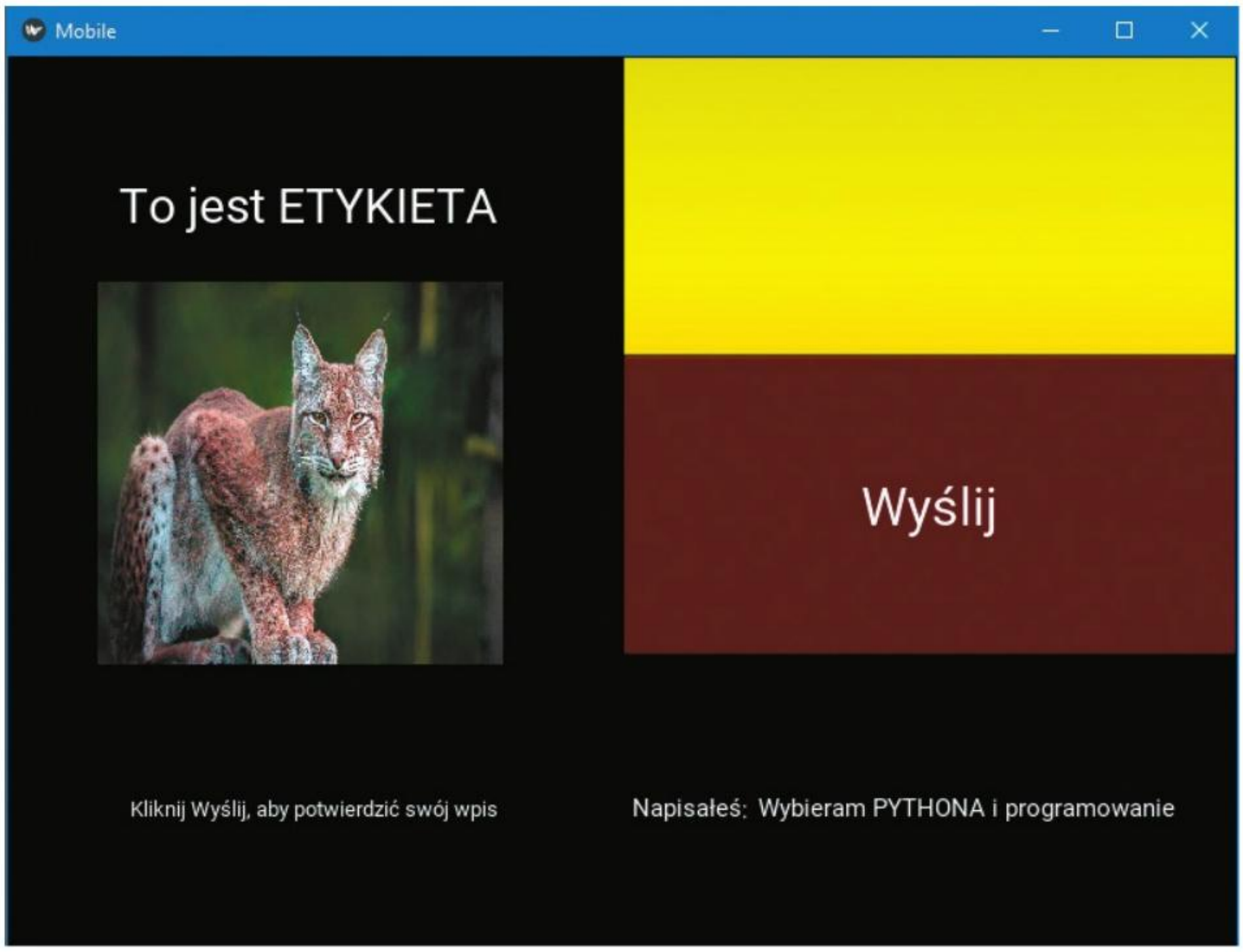
Dysponujemy już podstawą działającej, ale jednak statycznej aplikacji. Możemy wpisać tekst w element powstały z TextInput. Teraz go przechwycimy i wyświetlimy w innym miejscu.

W żółtym polu wpisujemy tekst: „Wybieram PYTHONA i programowanie".



Rys. 16.13. Widok uruchomionej aplikacji i wpisu w pole tekstowe

Po wysłaniu wpisanego tekstu (w prawym żółtym polu) został on wyświetlony na dole prawej kolumny.



Rys. 16.14. Widok uruchomionej aplikacji po kliknięciu przycisku Wyślij – wpis z pola tekstowego znika z żółtego pola i zostaje wyświetlony na dole prawej kolumny

Aby osiągnąć efekt przedstawiony na rys. 16.14, trzeba wprowadzić zmiany w kodzie.

- **Obsługa zdarzenia – naciśnięcie przycisku Button:**

```
self.submit.bind(on_press=self.nacisnij)
```

Po naciśnięciu przycisku (kod: on_press) zostanie wywołana funkcja o nazwie 'naciśnij'. Trzeba ją zaprogramować w kolejnym kroku.

- **Zaprogramowanie funkcji 'naciśnij':**

```
def nacisnij(self, instance): ...
```

- **Dodanie elementu w lewej kolumnie, informującego o potrzebie naciśnięcia przycisku Wyślij:**

```
self.label = self.add_widget(Label(text="Kliknij Wyślij,
aby potwierdzić twój wpis",font_size ="13sp"))
```

Na rys. 16.15 i 16.16 możesz zobaczyć, jak wyglądają zmiany w przygotowanym kodzie.

```
# Przycisk do wysyłania (Submit Button)
self.submit = Button(text="Wyślij", font_size=32)
self.submit.background_color = [1, 0, 0, 1]

# Dodanie obsługi kliknięcia (czyli zdarzenia)
do przycisku - po kliknięciu zostanie wywołana
funkcja o nazwie wypiszTekst
self.submit.bind(on_press=self.nacisnij)

# Dodaj widget - przycisk do wysyłania (Submit
Button)
self.add_widget(self.submit)

# Dodaj widget z Label (etykietą)
self.label = self.add_widget(Label(text="Kliknij
Wyślij, aby potwierdzić Twój wpis",font_size
="13sp"))
```

Rys. 16.15. Widok kodu aplikacji wykonanej w Kivy – po zmianach (od przycisku Button w dół pliku)

```
# Funkcja obsługująca kliknięcie
def nacisnij(self, instance):
    # przechwyć wpisany tekst do Inputa do nowej
    zmiennej
    namePrzechwycone = self.name.text

    # stworzymy nowy element strony Label, w którym
    wyświetlimy przechwycony tekst (ze zmiennej
    namePrzechwycone)
    self.add_widget(Label(text=f'Napisałeś:
    {namePrzechwycone}'))

    # Dodajmy jeszcze jedną funkcjonalność -
    wyczyszczenie pola Input po przechwyceniu
    wiadomości
    self.name.text = ""
```

Rys. 16.16. Widok kodu aplikacji wykonanej w Kivy po zmianach – dodanie funkcji **`nacisnij()`**

Prosta aplikacja mobilna, przygotowana w Kivy i działającą w środowisku Pythona, jest teraz w pełni funkcjonalna.

Jeżeli ma ona zadziałać na telefonie w systemie operacyjnym Android, trzeba ją „przerobić" (buildowanie) do paczki androidowej.

16.5. Język kv – projektowanie graficzne aplikacji

Tworzenie aplikacji internetowych, mobilnych i desktopowych ma wiele wspólnych elementów niemal w każdym ze środowisk informatycznych. Jednym z nich jest oddzielenie projektowania graficznego od właściwego kodu programistycznego. Poniżej kilka przykładów z różnych dziedzin tworzenia aplikacji.

Gdy przygotowujesz aplikację webową:

- opracowujesz jej projekt graficzny (oraz szkielet DOM) w językach programowania HTML i CSS (więcej informacji znajdziesz na stronie: https://www.w3.org/standards/webdesign/htmlcss);
- funkcjonalność dodajesz w językach skryptowych, np. JavaScript (więcej informacji: https://developer.mozilla.org/en-US/docs/Web/javascript) i PHP (więcej informacji: https://www.php.net/).

Podobnie działa zestaw:

- projekt graficzny – język XAML (więcej informacji znajdziesz na stronie: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/fundamentals/xaml);
- programowanie funkcjonalności – język C# (więcej informacji znajdziesz na stronie: https://docs.microsoft.com/pl-pl/dotnet/csharp/).

Takie samo podejście znajdziesz we frameworku Kivy:

- projekt graficzny – język kv;
- programowanie funkcjonalności – język Python 3.x.

Język kv

Użycie języka kv, nazywanego też kvlang lub język kivy, pozwala na oddzielenie budowy interfejsu graficznego od programowania zdarzeń (funkcjonalności) w języku Python (opierając się na specyfice frameworka Kivy). Umożliwia on[2]:

- tworzenie drzewa widżetów (struktury graficznej aplikacji) w sposób deklaratywny;
- wiązanie właściwości widżetów ze sobą i z wywołaniami zwrotnymi;
- szybkie prototypowanie;
- zwinne zmiany w interfejsie użytkownika (filozofia Agile).

Język kv ułatwia i przyspiesza pracę nad budową aplikacji i późniejszymi zmianami. Użycie kv jest wygodniejsze, a projekt zbudowany z jego pomocą – przejrzysty i zrozumiały. Plik języka kv oznacza się rozszerzeniem .kv.

Stwórz plik o nazwie testkv.kv, następnie przepisz do niego kod zawarty na rys. 16.17 – 16.18.

---

[2] Informacje opracowane na podstawie: https://kivy.org/doc/stable/guide/lang.html#concept-behind-the-language (dostęp: 30.03.2021).

```
#:kivy 2.0.0

<MyGridLayout>:

    GridLayout:
        cols:1
        size: root.width, root.height

        GridLayout:
            cols:2

            Label:
                text: "Imię"
            TextInput:
                id: imie
                multiline:False

            Label:
                text: "Nazwisko"
            TextInput:
                id: nazwisko
                multiline:False
```

Rys. 16.17. Widok kodu pliku testkv.kv – górna część

```
            Label:
                text: "Numer Telefonu"
            TextInput:
                id: numerTelefonu
                multiline:False
                font_size: 32

        Button:
            text: "Wyślij"
            font_size: 32
            background_color: (1,0,0,1)
```

Rys. 16.18. Widok kodu pliku testkv.kv – dolna część

Podobnie jak w Pythonie, wcięcia budują określoną strukturę widoku strony internetowej. Gdy zmienisz poziom wcięcia, czyli odległość od lewej krawędzi, kod przestanie działać.

Teraz trzeba zbudować za pomocą Pythona funkcjonalności uruchamiające budowaną aplikację. W tym samym katalogu stwórz plik testkv.py, a następnie przepisz do niego kod przedstawiony na rys. 16.19.

```
from kivy.app import App
from kivy.uix.widget import Widget
# import modułu do obsługi właściwości obiektu
from kivy.properties import ObjectProperty
# import modułu do obsługi plików .kv
from kivy.lang import Builder

Builder.load_file('testkv.kv')

class MyGridLayout(Widget):

    imie = ObjectProperty(None)
    nazwisko = ObjectProperty(None)
    numerTelefonu = ObjectProperty(None)

# Gdybyśmy nie użyli Buildera (zaimportowany), to i
tak aplikacja mogłaby widzieć plik .kv, ale trzeba by
wykonać nazewnictwo w odpowiedni sposób: przy nazwie
pliku kv nazwany test5 - poniższa klasa musiałaby się
nazywać Test5App (czyli złączenie - Test5+App)
class TestApp(App):
    def build(self):
        return MyGridLayout()

if __name__ == '__main__':
    TestApp().run()
```

Rys. 16.19. Widok kodu pliku testkv.py

Warto podkreślić znaczenie poniższych linijek kodu:

```
from kivy.lang import Builder
```

i

```
Builder.load_file('testkv.kv')
```

Pierwsza z nich importuje moduł Buildera, który do pliku 'testkv.py' ładuje plik 'testkv.kv'. Jest kilka możliwości załadowania zawartości kodu .kv. Prawdopodobnie najczęściej wykorzystuje się tę zastosowaną za pomocą Buildera. Drugą opisano na rys. 16.19.

Gdybyśmy nie użyli Buildera (zaimportowanego), to i tak aplikacja mogłaby widzieć plik .kv, ale trzeba by było przygotować nazewnictwo w odpowiedni sposób: przy pliku kv o nazwie test5 poniższa klasa musiałaby się nazywać Test5App.

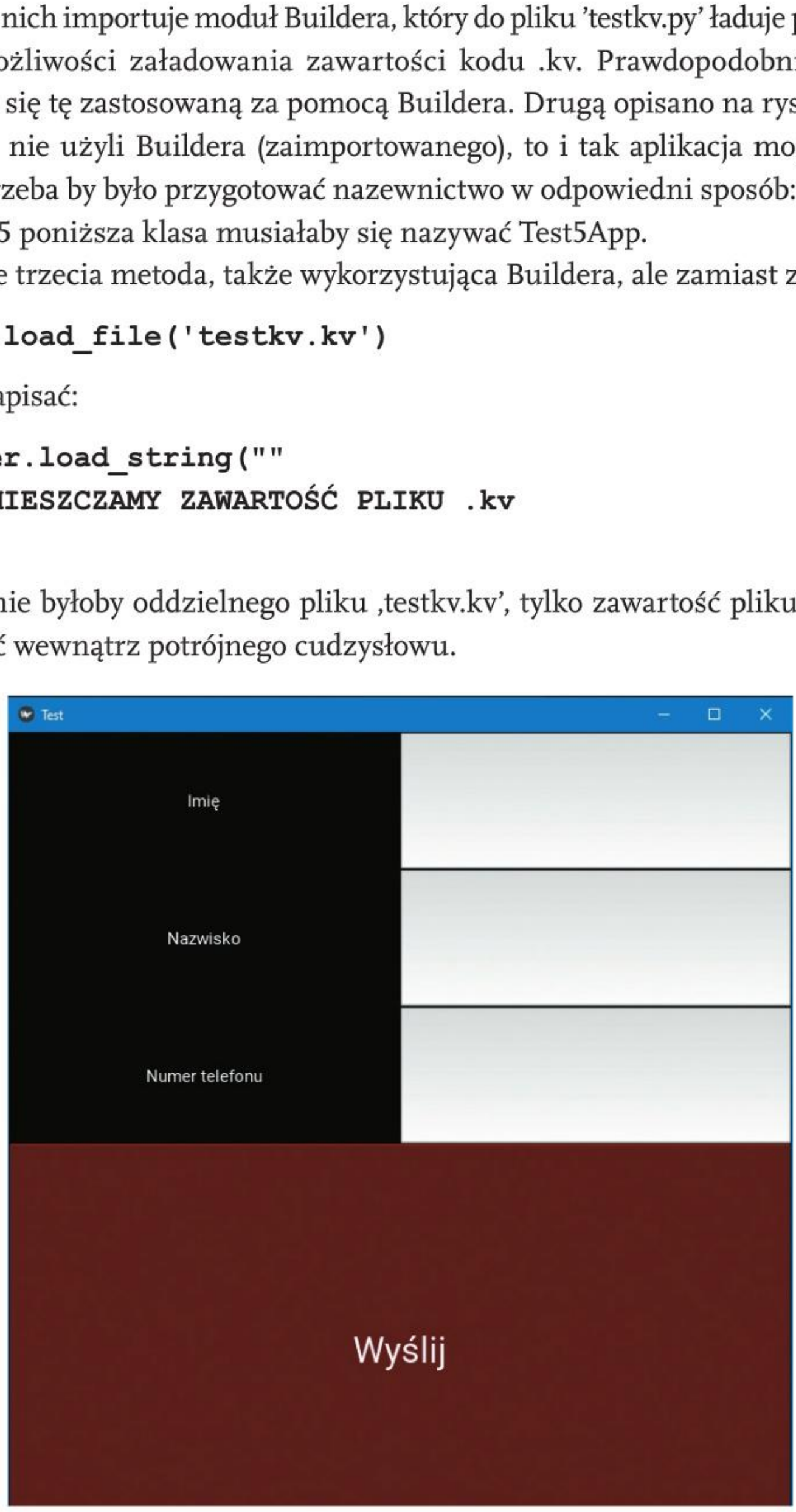
Jest jeszcze trzecia metoda, także wykorzystująca Buildera, ale zamiast zapisu:

```
Builder.load_file('testkv.kv')
```

należałoby napisać:

```
# Builder.load_string(""
TUTAJ UMIESZCZAMY ZAWARTOŚĆ PLIKU .kv
""")
```

Wówczas nie byłoby oddzielnego pliku ‚testkv.kv', tylko zawartość pliku .kv trzeba by było umieścić wewnątrz potrójnego cudzysłowu.



Rys. 16.20. Widok aplikacji stworzonej w kodach plików testkv.py i testkv.kv

Accordion – elementu UI

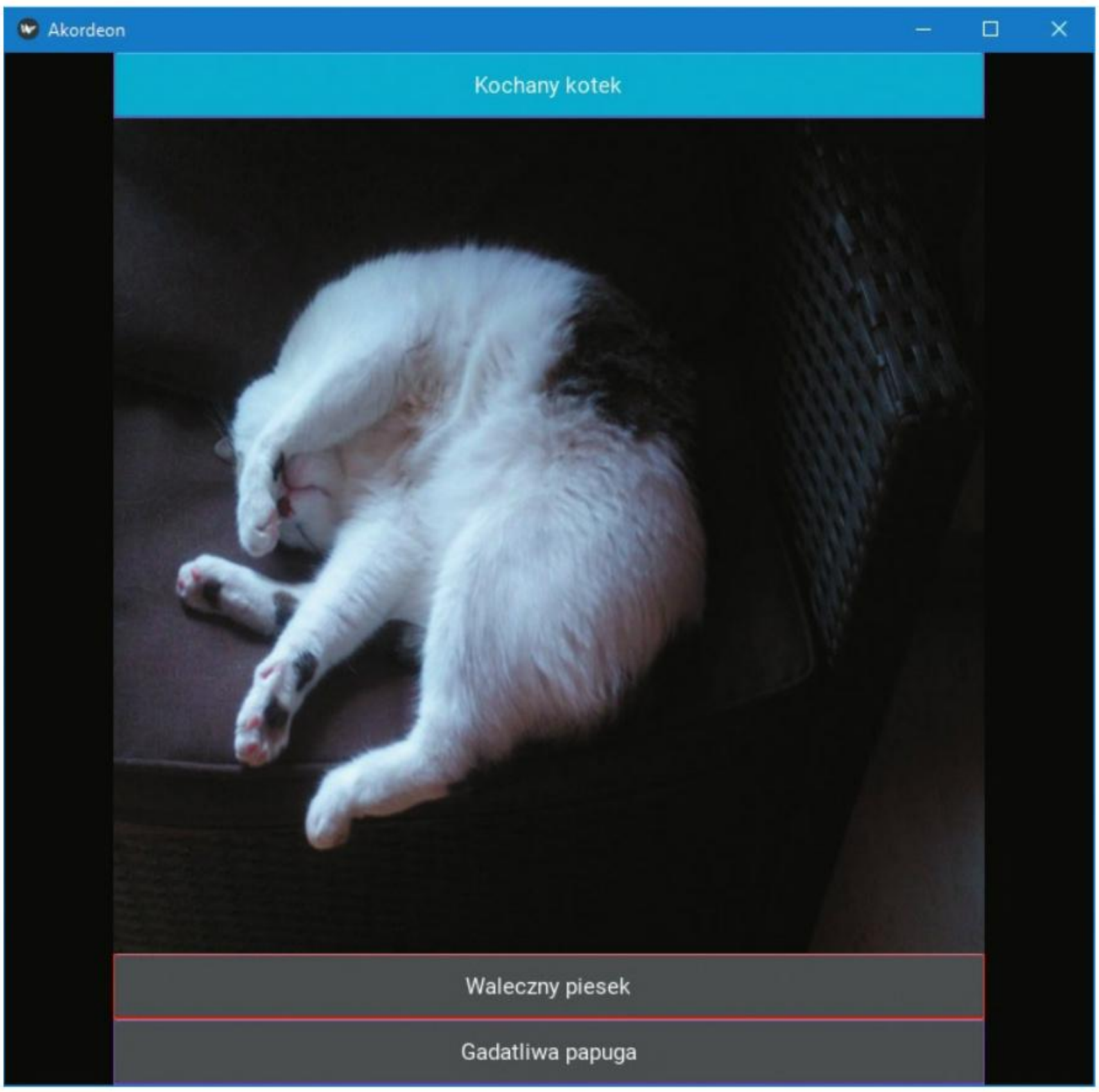
Jest to interaktywny zestaw przycisków, z których każdy po jego naciśnięciu oferuje funkcję rozwinięcia ukrytego obszaru znajdującego się pod nim. Accordion może być wykorzystany zarówno do stworzenia rozwijanego menu, jak i do wyświetlania zawartości, która ze względu na małą ilość miejsca na ekranie urządzenia mobilnego jest ukryta. Można go zastosować zarówno w ułożeniu pionowym, jak i poziomym.

W przypadku polecenia:

```
<Akordeon>:
orientation: 'vertical'
```

poszczególne przyciski będą układały się jeden pod drugim. Mamy do czynienia z przyjemną dla oka animacją elementów Accordion. Po naciśnięciu jednego z nich pozostałe przyciski przesuwają się w dół, a w wolnym obszarze pokazuje się zdjęcie.

Efekt działania aplikacji przedstawiono na rys. 16.21. Na następnej stronie opiszemy kody. Zaczniemy od pliku Pythona, odpowiadającego za funkcjonalność aplikacji.



Rys. 16.21. Widok aplikacji mobilnej wykorzystującej element UI frameworka Kivy – Accordion

```
import kivy
from kivy.app import App
kivy.require('2.0.0')

# Widżet Accordion jest formą menu, w którym
opcje są ułożone w pionie (jeden na drugim) lub
w poziomie (obok siebie), a po dotknięciu danego
elementu - widżet otwiera się, aby wyświetlić
jego zawartość.
from kivy.uix.accordion import Accordion

# Stworzenie klasy Accordion, która wykonywana
jest w pliku .kv
class Akordeon(Accordion):
    pass

# Stworzenie klasy App
class AkordeonApp(App):
    def build(self):
        return Akordeon()

# Uruchomienie Aplikacji
if __name__ == '__main__':
    AkordeonApp().run()
```

Rys. 16.22. Widok kodów napisanych w języku programowania Python – aplikacja wykorzystująca element UI frameworka Kivy Accordion

Accordion jest jednym z ciekawszych elementów UI frameworka Kivy. Możesz zobaczyć jego działanie w ramach zainstalowanego `kivy_examples`. Znajdziesz tam także mnóstwo pokazowych składowych frameworka, które w łatwy sposób można zainstalować wewnątrz tworzonej aplikacji mobilnej. Jeżeli klikniesz w obrazek robaczka (będąc w `kivy_examples`), to ujrzysz kod języka kv, tworzący dany element UI Kivy.

```
#:kivy 2.0.0

# Ustawienia obrazka, który zostanie w
jednym z kolejnych wierszy
zaimportowany do projektu
<MyImage@Image>:
    keep_ratio: False
    allow_stretch: True

<AccordionItem>:
    font_size: 92
    # background_normal: ''
    background_color: (1,0,0,1)

<Akordeon>:
    orientation: 'vertical'
    pos_hint: {'center_x':0.5}
    size_hint: (0.8,1)
    # padding: 50
```

Rys. 16.23. Widok kodów napisanych w języku programowania kv – aplikacja wykorzystująca element UI frameworka Kivy Accordion (część 1)

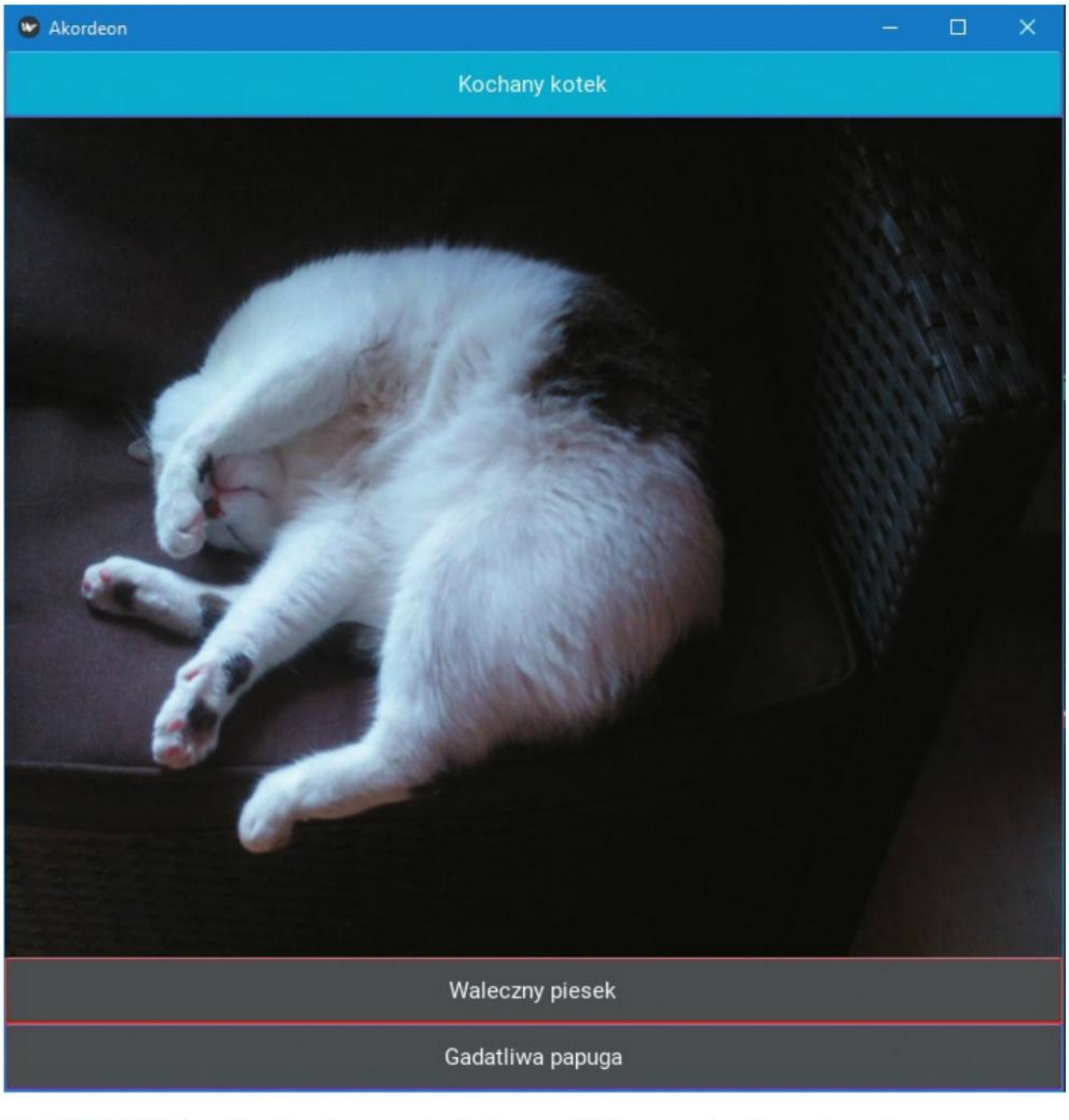
Warto zwrócić uwagę na elementy kodu:

```
<MyImage@Image>:
    keep_ratio: False
    allow_stretch: True
```

Określa on podstawowe ustawienia obrazka, który zostanie zaimportowany w jednym z kolejnych wierszy do projektu:

```
pos_hint: {'center_x':0.5}
size_hint: (0.8,1)
```

Dzięki niemu widget Accordion nie rozciąga się na całą szerokość strony. Po prawej i po lewej stronie znajdują się „puste obszary”. Jeżeli zakomentujesz lub usuniesz powyższy fragment kodu języka kv, to uzyskasz efekt dopasowania elementu Accordion na całą szerokość aplikacji (strony), tak jak na rys. 16.24.



Rys. 16.24. Widok aplikacji wykorzystującej element UI frameworka Kivy – Accordion

Na rys. 16.25 i 16.26 przedstawiono kolejne części kodu pliku .kv, który zapisano językiem frameworka Kivy.

Nawiązując do wskazanych na poprzedniej stronie ustawień obrazka, tutaj będziemy mieli kontynuację logiczną tamtego kodu, czyli dodanie obrazka spod określonego adresu. W naszym przypadku będzie nim katalog projektu. Co ważne, mógłby to być inny katalog, a także adres internetowy, z którego obrazek byłby załadowany. Kod ustawiający adres jest bardzo prosty:

```
MyImage:
    source: 'kotekLapka.jpg'
```

```
AccordionItem:
    title: 'Waleczny piesek'
    background_color: (1,0,0,1)
    canvas.before:
        Color:
            rgba: self.background_color
        Rectangle:
            size: self.size
            pos: self.pos
    MyImage:
        source: 'piesek.jpg'

AccordionItem:
    title: 'Gadatliwa papuga'
    background_color: (185/255,0,1,1)
    color: (185/255,0,1,1)
    canvas.before:
        Color:
            rgba: self.background_color
        Color:
            rgba: self.color
        Rectangle:
            size: self.size
            pos: self.pos
    MyImage:
        source: 'piesek.jpg'
```

Rys. 16.25. Widok kodów napisanych w języku programowania kv – aplikacja wykorzystująca element UI frameworka Kivy Accordion (część 2)

```
AccordionItem:
    title: 'Gadatliwa papuga'
    background_color: (185/255,0,1,1)
    color: (185/255,0,1,1)
    canvas.before:
        Color:
            rgba: self.background_color
            rgba: self.color
        Rectangle:
            size: self.size
            pos: self.pos
MyImage:
    source: 'piesek.jpg'
```

Rys. 16.26. Widok kodów napisanych w języku programowania kv – aplikacja wykorzystująca element UI frameworka Kivy Accordion (część 3)

16.6. Projekt aplikacji – Stoper

W tym rozdziale wykonamy nieco bardziej zaawansowaną aplikację – Stoper. Będzie można go włączać i wyłączać oraz restartować. Wszystko będzie się odbywało za pomocą przycisków. Dodatkowo podzielimy ekran aplikacji na więcej obszarów. Wybierzemy układ strony BoxLayout.

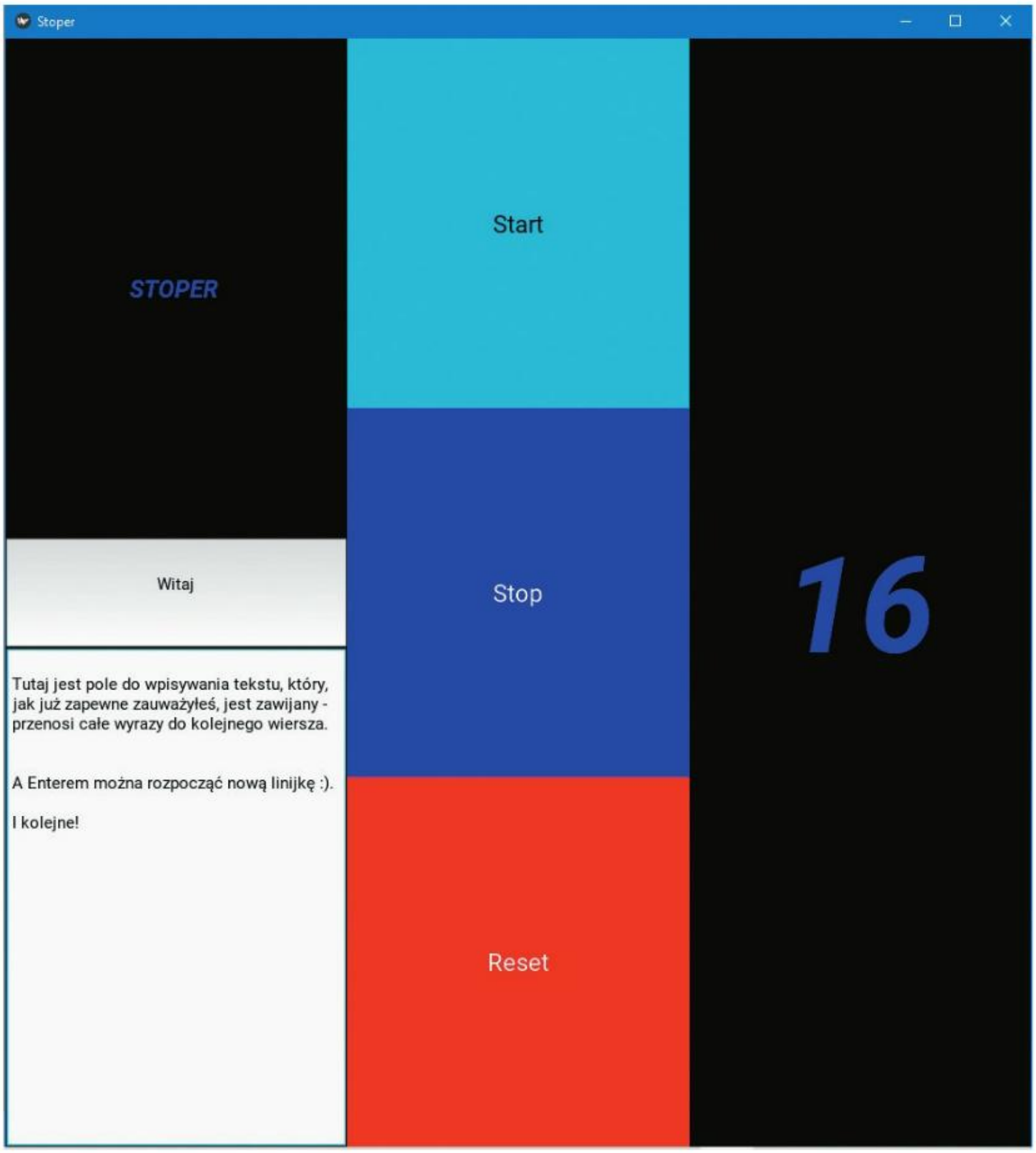
W lewej kolumnie dodamy trzy bloki: dwa informacyjne, a najniższy będzie edytowalny. Znajdzie się tam element UI Kivy o nazwie InputText.

Dzięki użyciu odpowiednich kodów w pliku .kv będzie można wpisywać do niego tekst zawijany – całe wyrazy będą przenoszone do kolejnej linii. Będzie można także przyciskiem Enter przejść do nowej linii.

Zwróć uwagę na zakodowanie kolorów za pomocą popularnego w grafice komputerowej standardu RGBA (R – red, czerwony, G – green, zielony, B – blue, niebieski, A – alpha, kanał alfa):

```
background_color: (1,0,0,1).
```

Na rys. 16.27 przedstawiono efekt kodowania opisanej funkcjonalności.



Rys. 16.27. Widok aplikacji Stoper napisanej we frameworku Kivy (języku programowania kv i Python)

Na rys. 16.27 przedstawiono widok aplikacji Stoper. Po jej uruchomieniu sekundy są liczone i pokazywane w skrajnej prawej kolumnie. Gdy naciśniesz przycisk:

- Start – rozpoczniesz kontynuację odliczania, czyli jeżeli np. stan stopera wynosił 43, to po kliknięciu tego przycisku Stoper pokaże po sekundzie liczbę 44;
- Stop – zatrzymasz odliczanie;
- Reset – zresetujesz stoper do ustawionej wartości. W przedstawionych kodach ustawiono tę wartość na 200, czyli ta liczba wyświetli się w czarnej prawej kolumnie po naciśnięciu przycisku Reset.

Aby zaprogramować projekt, przygotuj dwa pliki:

- timerApp.py,
- timerApp.kv.

Do pierwszego z nich wpisz kody języka programowania Python zawarte na grafikach na następnych stronach. Drugi omówimy w dalszej kolejności.

```
from kivy.app import App
from kivy.lang import Builder
from kivy.properties import NumericProperty
# BoxLayout umożliwia ustawienie elementów
potomnych w pionowym lub poziomym pudełku.
from kivy.uix.boxlayout import BoxLayout
# Obiekt Clock - zegar - umożliwia zaplanowanie
wywołania funkcji w określonym czasie (przyszłości)
from kivy.clock import Clock

Builder.load_file("timerApp.kv")
```

Rys. 16.28. Widok kodów aplikacji Stoper – framework Kivy, plik timerApp.py (część 1)

```
# Tworzymy układ klasy
class Stoper(BoxLayout):

    stan_stopera = NumericProperty()

    def __init__(self, **kwargs):
        # Funkcja wbudowana super() zwraca obiekt
        proxy, który daje możliwość odwoływania się
        do klasy nadrzędnej poprzez „super”.
        super(Stoper, self).__init__(**kwargs)

        # Tworzymy zegar i dokonujemy powiększania
        czasu o 1, czyli 1 sekundę.
        Clock.schedule_interval(self.
        increment_time, .1)

        self.increment_time(0)

    # Funkcja do zwiększania czasu o sekundę
    def increment_time(self, interval):
        self.stan_stopera += .1
```

Rys. 16.29. Widok kodów aplikacji Stoper – framework Kivy, plik timerApp.py (część 2)

```
    # Funkcja rozpoczynająca liczenie
    def start(self):

        # Na bazie zaimportowanego obiektu Clock
        z biblioteki Kivy uruchamiamy obliczanie
        Clock.unschedule(self.increment_time)
        Clock.schedule_interval(self.
        increment_time, .1)

    # Funkcja zatrzymująca liczenie
    def stop(self):
        Clock.unschedule(self.increment_time)

# Utworzenie klasy App
class StoperApp(App):
    def build(self):
        return Stoper()

# Uruchomienie aplikacji
StoperApp().run()
```

Rys. 16.30. Widok kodów aplikacji Stoper – framework Kivy, plik timerApp.py (część 3)

Teraz trzeba umieścić kody w pliku timerApp.kv. Pamiętaj, aby oba pliki znajdowały się w tym samym katalogu.

CIEKAWOSTKA

Framework Kivy dostarcza programistom aplikacji mobilnych wiele bardzo przydatnych rozwiązań, m.in. obiekt kivy.clock, który (jak można przeczytać w dokumentacji frameworka): „umożliwia planowanie wywołania funkcji w przyszłości raz lub wielokrotnie w określonych odstępach".
Więcej na ten temat można znaleźć na oficjalnej stronie omawianej aplikacji: https://kivy.org/doc/stable/api-kivy.clock.html#.

```
#:kivy 2.0.0

<Stoper>:

    # Przypisywanie pożądanego wyrównania i wyglądu
    do pól informacyjnych zawartych w pierwszej
    kolumnie
    BoxLayout:
        orientation: 'vertical'

        Label:
            text: "STOPER"
            halign: 'center'
            valign: 'middle'
            multiline:True
            font_size: 22
            color: (0,0,1,1)
            bold: True
            italic: True
            outline_color: (0,0,0)
            outline_width: 33
```

Rys. 16.31. Widok kodów aplikacji Stoper – framework Kivy, plik timerApp.kv (część 1)

Zmiana tempa czasu wyświetlania kolejnych liczb

Po przepisaniu kodów do plików timerApp.py i timerApp.kv warto wyszukać w kodzie linijki zawierające:

```
Clock.schedule_interval(self.increment_time, .1).
```

Powinny być dwie. Zmień zapis z `.1` na `.3`. Najpierw zrób tak dla pierwszej z linijek, zapisz kody i wywołaj aplikację. Następnie wycofaj zmiany i zrób to samo z drugą ze wskazywanych linijek kodu.

```
        TextInput:
            text: "Witaj"
            pos_hint: {'center_x': 0.5, 'center_y':
            .95}
            size_hint_y: None
            halign: "center"
            valign: "middle"
            padding:33

        TextInput:
            hint_text:'Enter text'
```

Rys. 16.32. Widok kodów aplikacji Stoper – framework Kivy, plik timerApp.kv (część 2)

```
    # Przypisanie ułożenia na stronie dla
    przycisków: Start, Stop, Reset
    BoxLayout:
        orientation: 'vertical'

        Button:
            text: 'Start'
            on_press: root.start()
            font_size: 22
            background_normal: ''
            background_color: (0,200/255,1)
            color: (0,0,0,1)

        Button:
            text: 'Stop'
            on_press: root.stop()
            font_size: 22
            background_normal: ''
            background_color: (0,0,1,1)
```

Rys. 16.33. Widok kodów aplikacji Stoper – framework Kivy, plik timerApp.kv (część 3)

```
    Button:
        text: 'Reset'
        # Zmienna stan_stopera
        on_press: root.stan_stopera = 200
        font_size: 22
        background_normal: ''
        background_color: (1,0,0,1)

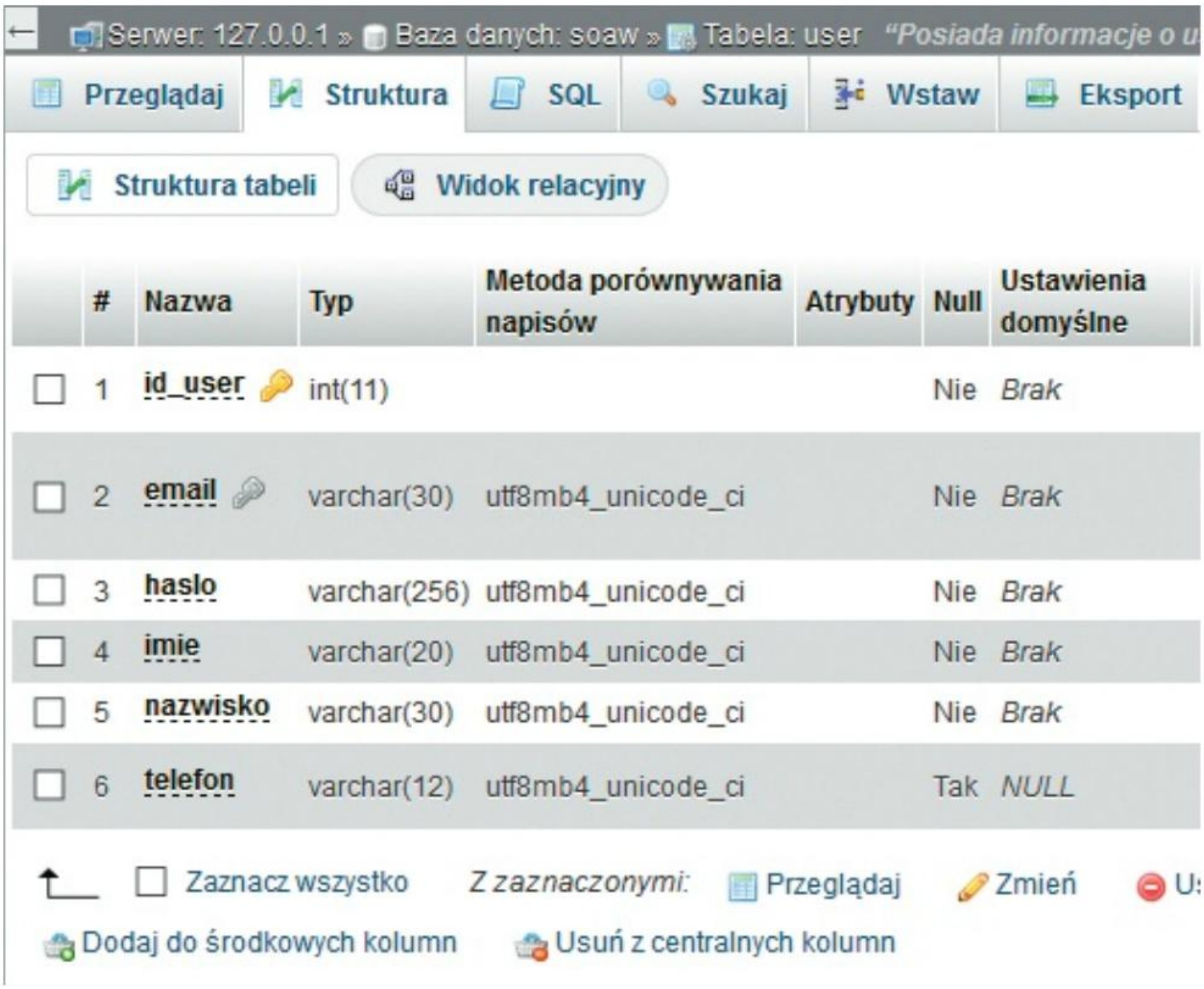
# Label - etykieta wyświetlająca stan stopera,
czyli odliczanie
Label:
    text: str(round(root.stan_stopera))
    text_size: self.size
    halign: 'center'
    valign: 'middle'
    font_size: 122
    color: (0,0,1,1)
    bold: True
    italic: True
    outline_color: (0,0,0)
    outline_width: 33
```

Rys. 16.34. Widok kodów aplikacji Stoper – framework Kivy, plik timerApp.kv (część 4)

16.7. Współpraca Kivy z relacyjną bazą danych SQL

W tym rozdziale zajmiemy się połączeniem frontendu z backendem – aplikacji stworzonej w Kivy z bazą danych osadzoną wewnątrz danego środowiska serwerowego.

Aby wykonać zadania w tym rozdziale, musisz mieć zainstalowanego i włączonego XAMPP-a. Jeżeli nie masz aktualnej instalacji, możesz pobrać tę aplikację ze strony: https://www.apachefriends.org/pl/index.html. Wiedzę o phpMyAdmin najlepiej czerpać z oficjalnej strony projektu https://www.phpmyadmin.net/ (po zainstalowaniu aplikacji XAMPP dostęp do dokumentacji pod adresem http://localhost/phpmyadmin/doc/html/index.html). Specyfikację bazy danych MariaDB (w pakiecie XAMPP) znajdziesz pod adresem projektu https://mariadb.com/.



Rys. 16.35. phpMyAdmin – widok struktury przykładowej tabeli

Alternatywą dla pakietu XAMPP i zawartych w nim bazy MariaDB i phpMyAdmin może być poniższy zestaw:

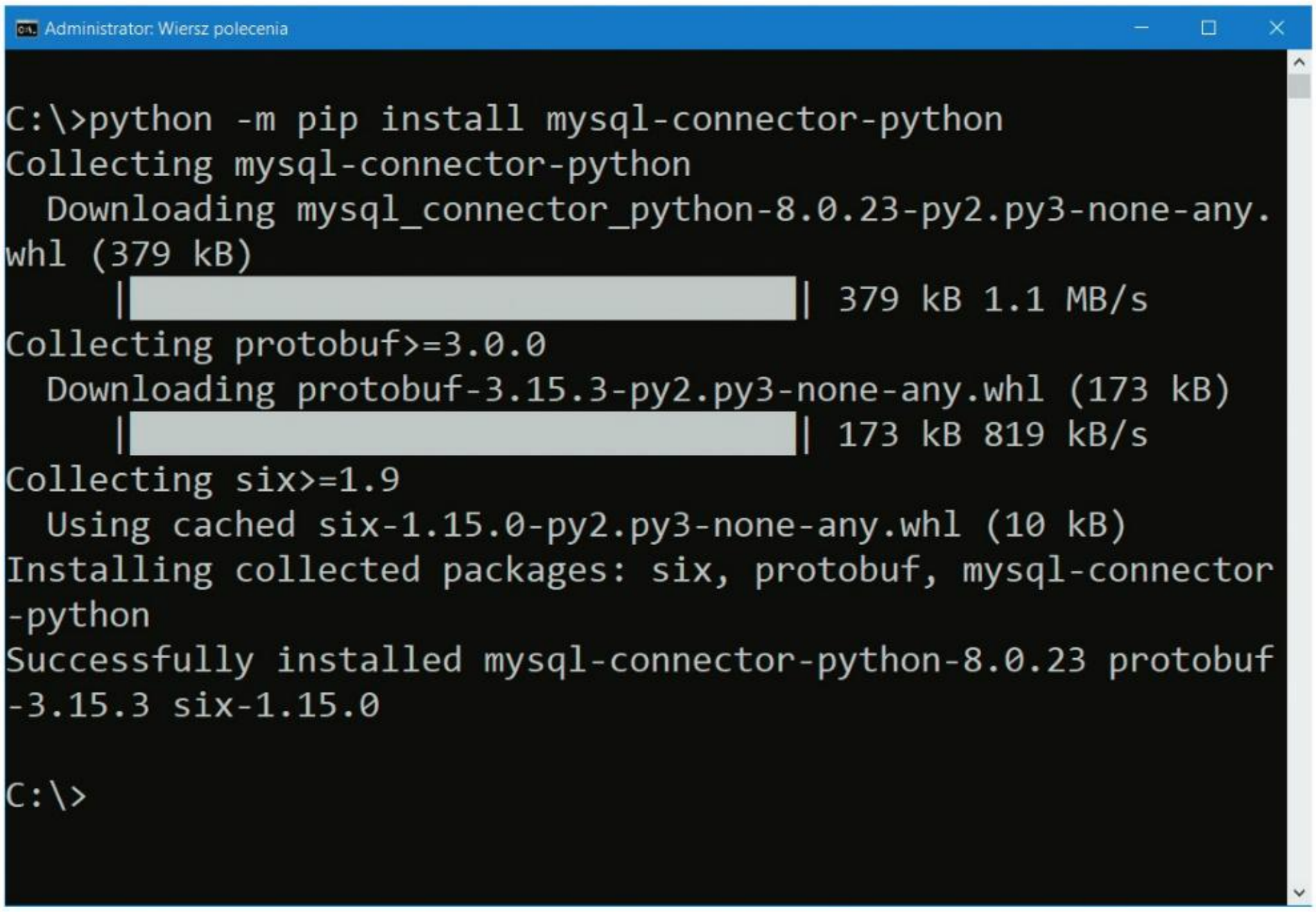
- SQLite – strona projektu: https://sqlite.org/index.html
  Jest to baza danych przechowywana jako plik z rozszerzeniem .db, dzięki czemu nie musi być utrzymywana na serwerze. Warto poszukać w internecie i zainstalować na swoim komputerze menedżera obsługującego tego typu bazy. Jednym z nich może być DB Browser for SQLite (strona domowa projektu: https://sqlitebrowser.org/). Ta bezpłatna aplikacja umożliwia przeglądanie plików stworzonych w technologii SQLite. Można ją pobrać ze wskazanego adresu. Istnieje w wersji instalacyjnej i jako program niepotrzebujący instalacji w systemie.

Przed przystąpieniem do pracy warto zainstalować w konsoli łącznik języka programowania Python z technologią – odpowiednio SQL (MariaDB) lub SQLi.

Na następnej stronie znajdziesz właściwe komendy.

SQL – instalacja łącznika języka programowania Python i relacyjnej bazy danych MariaDB:

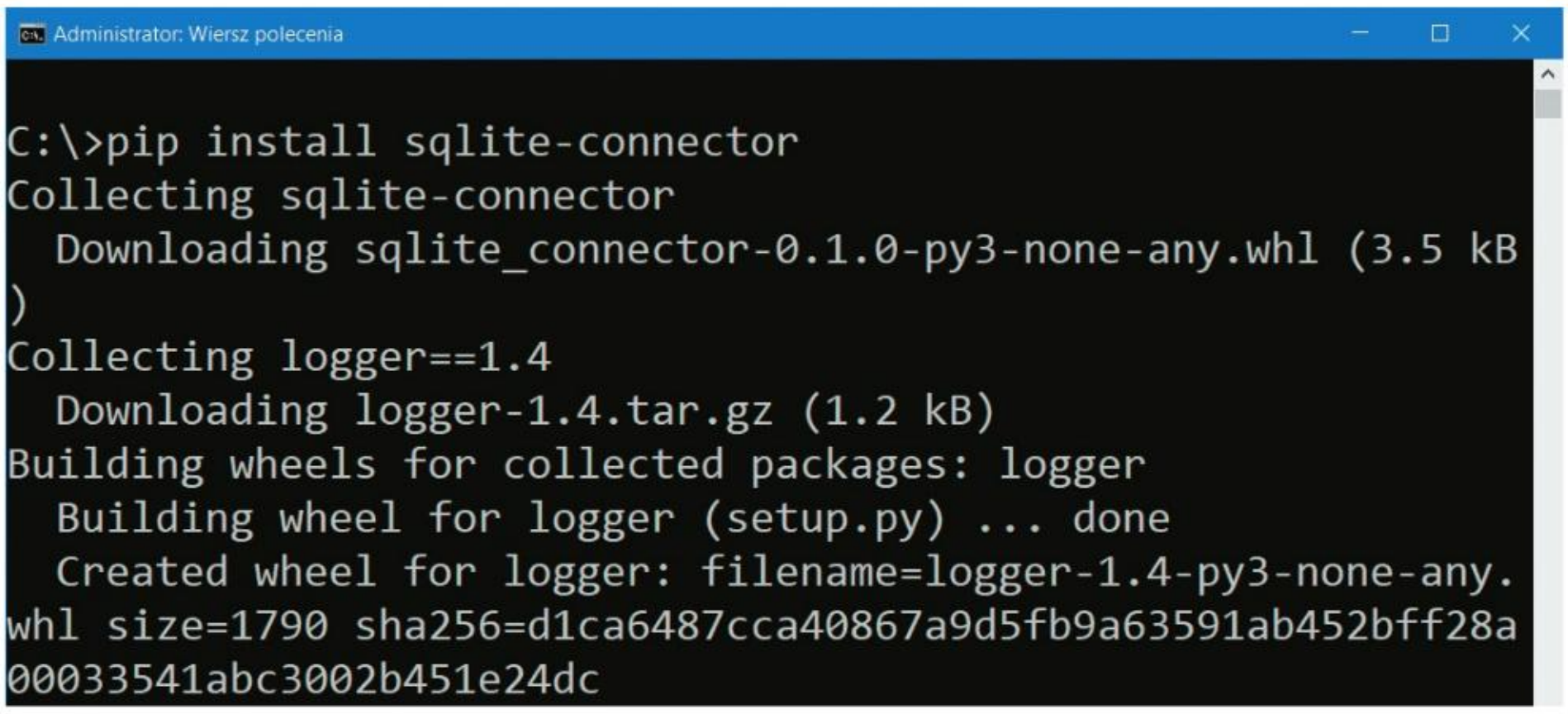
```
python -m pip install mysql-connector-python
```



Rys. 16.36. Instalacja modułu Pythona umożliwiającego połączenie z relacyjną bazą danych SQL

SQLi – instalacja łącznika języka programowania Python i relacyjnej bazy danych SQLi:

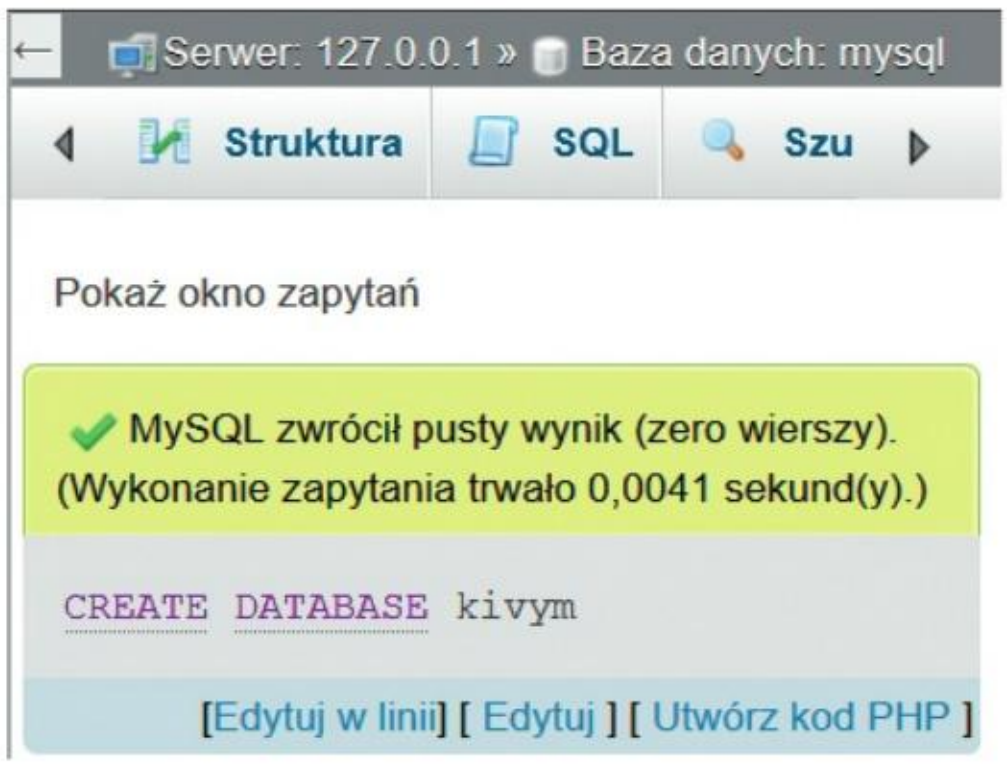
```
pip install sqlite-connector
```



Rys. 16.37. Instalacja modułu Pythona umożliwiającego połączenie z relacyjną bazą danych SQLi

Baza danych – tworzenie w XAMPP (MariaDB)

Rozpocznijmy od polecenia SQL tworzącego bazę danych. Możesz je wykonać wewnątrz pakietu XAMPP – w panelu phpMyAdmin: **`CREATE DATABASE kivym`**



Rys. 16.38. Potwierdzenie z panelu SQL phpMyAdmin stworzenia i dodania nowej relacyjnej bazy danych

Alternatywnym rozwiązaniem dla programisty jest stworzenie bazy bezpośrednio w pliku Pythona (rys. 16.39).

timerApp.py | sqlite.py | sqlitem1.py

naukaPython > sql > sqlitem1.py > ...

```
1   import mysql.connector
2
3   baza = mysql.connector.connect(
4     host="localhost",
5     user="root",
6     password=""
7   )
8
9   poloczenie = baza.cursor()
10
11  try:
12    poloczenie.execute("CREATE DATABASE kivy")
13  except Exception:
14    print("Baza już istnieje")
15
16
17
```

Rys. 16.39. Kod napisany w językach programowania Python i SQL – dodanie nowej bazy danych do pakietu XAMPP (MariaDB)

```
TERMINAL ···  2: Python

PS C:\kivy> & "C:/Program Files/Python39/python.exe" c:/kivy/
naukaPython/sql/sqlitem1.py
Baza już istnieje
PS C:\kivy>
```

Rys. 16.39. Cd. Kod napisany w językach programowania Python i SQL – dodanie nowej bazy danych do pakietu XAMPP (MariaDB)

```
.py  sqlite.py  sqlitem1.py  sqlitem2.py ×
naukaPython > sql > sqlitem2.py > ...
1   import mysql.connector
2
3   baza = mysql.connector.connect(
4     host="localhost",
5     user="root",
6     password=""
7   )
8
9   poloczenie = baza.cursor()
10
11  createTable = "CREATE TABLE kivym.languages (id INT
    AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), ocena INT)"
12
13  insertInTo = "INSERT INTO kivym.languages Values (null,
    'Python', 5), (null, 'JavaScript', 5), (null, 'PHP', 5)"
14
15  try:
16    poloczenie.execute("CREATE DATABASE kivym")
17  except Exception:
18    poloczenie.execute(createTable)
19
```

```
TERMINAL ···  2: Python

PS C:\kivy> & "C:/Program Files/Python39/python.exe" c:/kivy/nauk
aPython/sql/sqlitem2.py
PS C:\kivy>
```

Rys. 16.40. Kod napisany w językach programowania Python i SQL – dodanie nowej tabeli language do bazy danych ‘kivym’

py sqlite.py sqlitem1.py sqlitem2.py

naukaPython > sql > sqlitem2.py > ...

```
1   import mysql.connector
2
3   baza = mysql.connector.connect(
4     host="localhost",
5     user="root",
6     password=""
7   )
8
9   poloczenie = baza.cursor()
10
11  createTable = "CREATE TABLE kivym.languages (id INT
    AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), ocena INT)"
12
13  insertInTo = "INSERT INTO kivym.languages VALUES (null,
    'Python', 5), (null, 'JavaScript', 5), (null, 'PHP', 5)"
14
15  try:
16    poloczenie.execute("CREATE DATABASE kivym")
17  except Exception:
18    poloczenie.execute(insertInTo)
19    baza.commit()
```

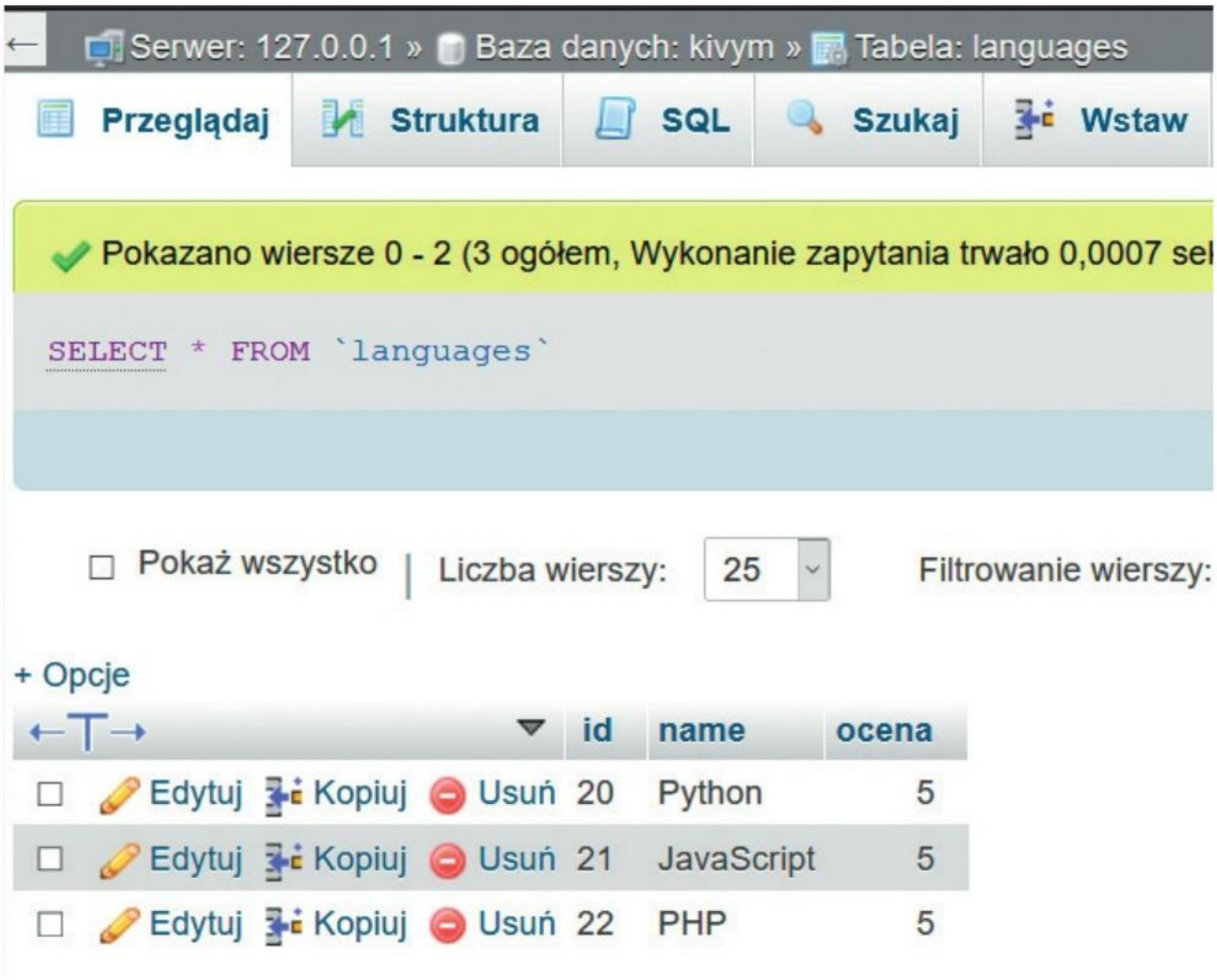
TERMINAL ... 3: Python

```
PS C:\kivy> & "C:/Program Files/Python39/python.exe" c:/kivy/nauka
Python/sql/sqlitem2.py
PS C:\kivy>
```

Rys. 16.41. Kod napisany w językach programowania Python i SQL – dodanie w tabeli language trzech wierszy z przykładowymi danymi

Wykorzystanie instrukcji **`try: except:`** nie jest konieczne. Istnieją inne, równie dobre rozwiązania. Alternatywnie zabezpieczenie przed błędem, który mógłby powstać w trakcie stworzenia tabeli już istniejącej, daje także język SQL. Dla przykładu moglibyśmy użyć:

```
createTable = "CREATE TABLE IF NOT EXISTS kivym.languages (id
INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), ocena INT)"
```



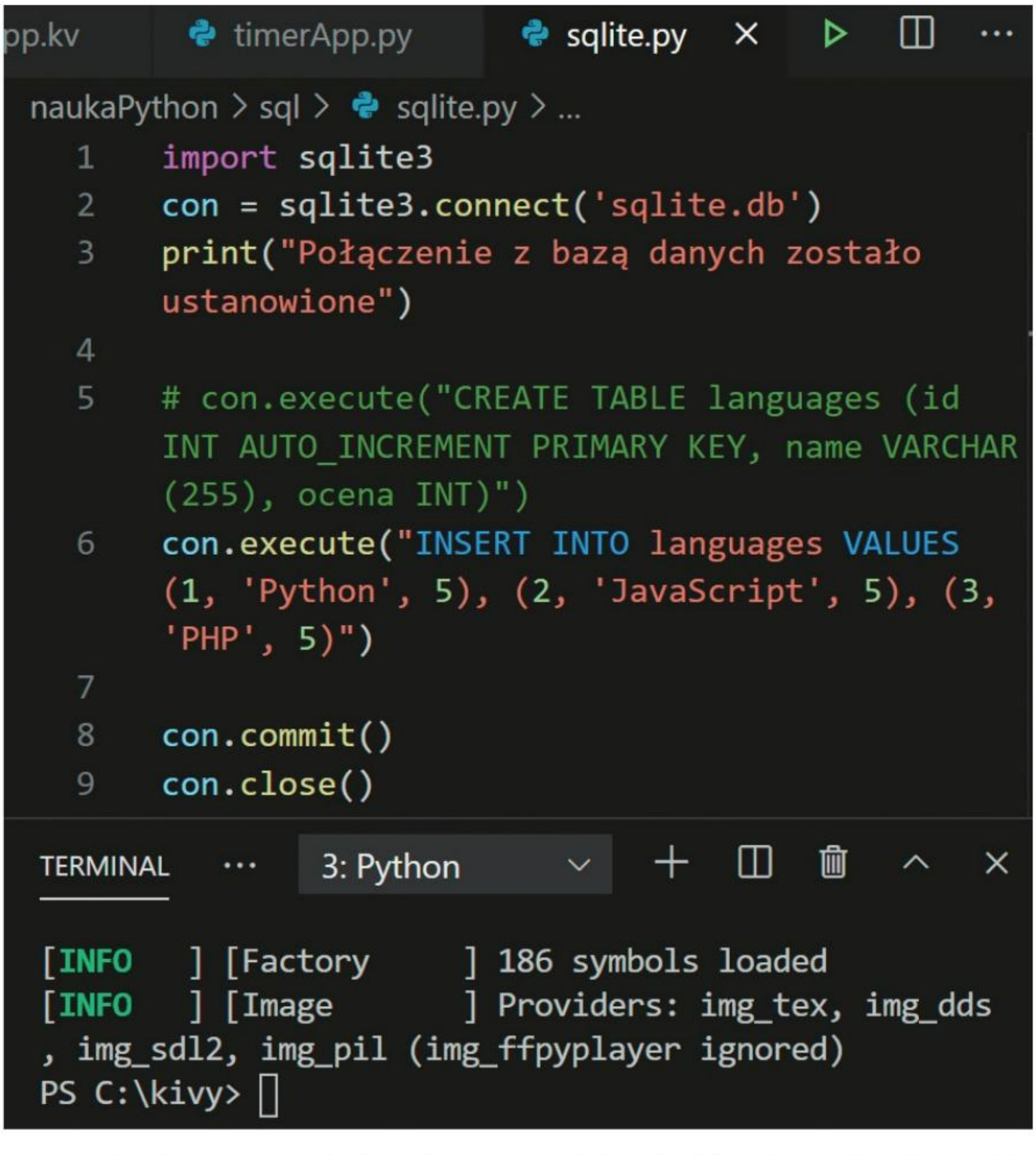
Rys. 16.42. Widok phpMyAdmin, trzy rekordy z przykładowymi danymi – efekt współdziałania Pythona i SQL

Baza danych – stworzenie na bazie SQLite

W przypadku SQLi bazą danych będzie plik z rozszerzeniem .db. Utwórz go i umieść w tym samym katalogu co plik sqlite.py, dzięki temu powstanie prosta ścieżka dostępu do wymienionego zasobu. W pliku Pythona dodaj kody takie jak na rys. 16.43.

Gdy już istnieje baza danych SQLite, to możesz do niej zajrzeć za pomocą wspomnianego menedżera baz – programu DB Browser for SQLite. Na rys. 16.44 przedstawiono wygląd programu od środka.

Python współpracuje również z wieloma innymi rodzajami baz danych. Za każdym razem, gdy chcesz korzystać z danego rodzaju bazy danych, musisz poszukać odpowiedniego łącznika (jak pokazywany wcześniej mysql-connector-python). Dzięki zastosowaniu takiego rozwiązania Python otrzyma moc współpracy z danym rodzajem bazy danych.



Rys. 16.43. Kod wykonany w językach Python i SQL – dodaje do pliku z bazą sqlite.db (przykładowe rekordy z danymi)

Na rys. 16.43 znajdują się dwie bardzo ważne informacje dla programisty Pythona w zakresie obsługi bazy danych SQLite:

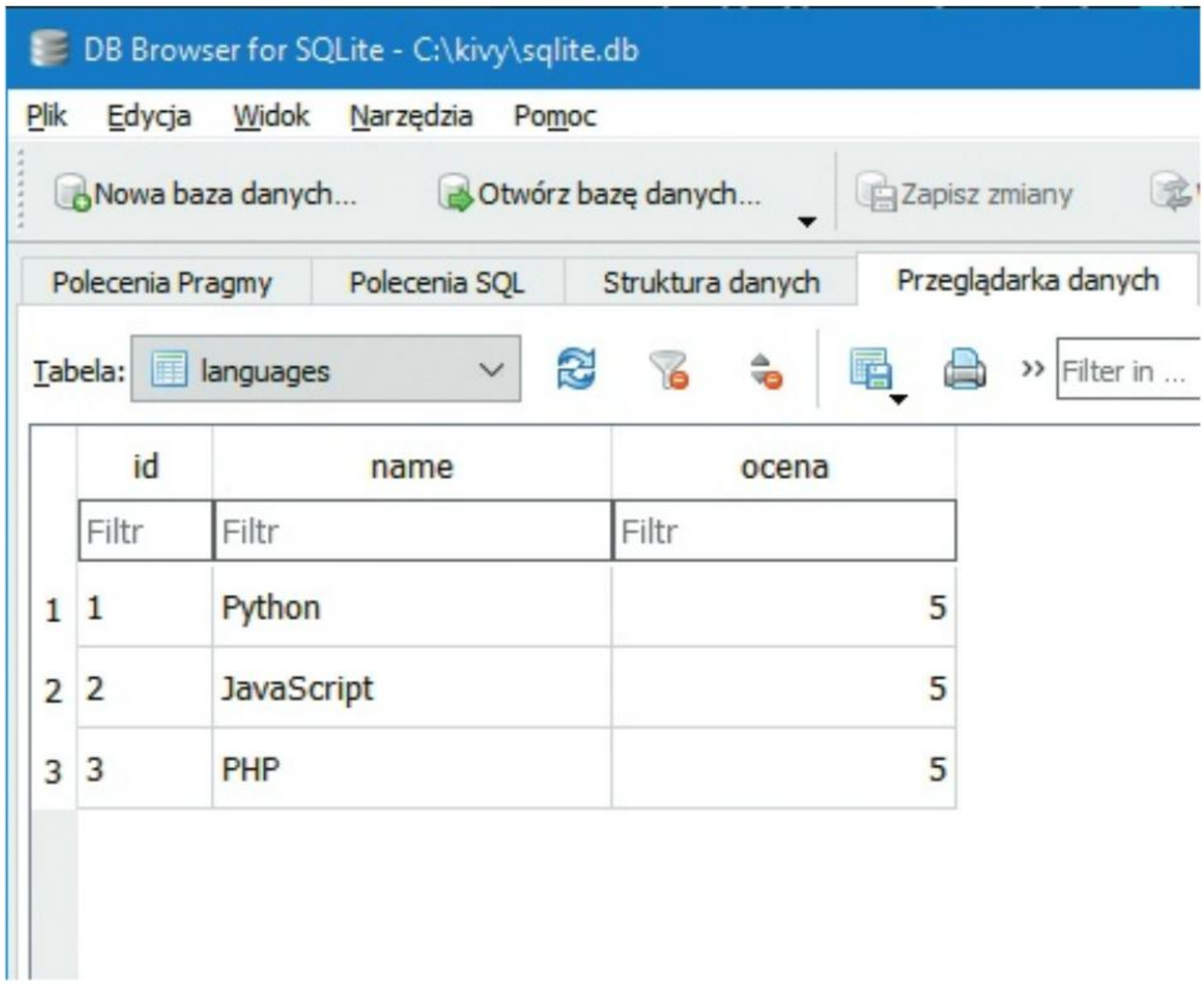
- zaimportowanie modułu do obsługi bazy danych typu SQLite (więcej na ten temat można znaleźć na oficjalnej stronie projektu Python: https://docs.python.org/3/library/sqlite3.html)

  ```
  import sqlite3
  ```

- utworzenie obiektu Connection, który reprezentuje bazę danych (więcej na ten temat można znaleźć w oficjalnej dokumentacji Pythona: https://docs.python.org/3/library/sqlite3.html#sqlite3.Connection)

  ```
  con = sqlite3.connect('sqlite.db')
  ```

  Dane będą przechowywane w pliku 'sqlite.db'.



Rys. 16.44. Widok menedżera baz danych SQLite, czyli programu DB Browser for SQLite – rekordy z przykładowymi danymi, efekt współdziałania Pythona i SQL

Aplikacja bazodanowa

W dalszej części podręcznika utworzymy aplikację, która będzie sprawdzała poprawność wpisanych danych z zasobami posiadanymi w bazie. Gdy wpiszesz właściwe dane, otrzymasz określoną informację, a jeżeli dane nie będą właściwe, program poinformuje cię o tym. Skupimy się na oprogramowaniu MySQL. Na zapleczu wykorzystamy aplikację XAMPP, czyli zawartą w niej bazę danych MariaDB, głównie dlatego, że z tej technologii korzystaliśmy w pierwszej części podręcznika podczas pracy nad rozwijaniem aplikacji webowej. W projektowaniu i rozwijaniu aplikacji mobilnych, szczególnie gdy programista ma do czynienia z prostą aplikacją, wygodniej jest korzystać z dobrodziejstw SQLi. Zauważ, że aby jej używać, nie trzeba włączać serwera ani nawet go mieć.

W rozdziale 16.4 stworzyliśmy kod przechwytujący z pola TextInput wpis użytkownika. Musimy wrócić do pliku z kodem tamtego projektu. W kodzie pliku wprowadzimy zmiany.

Zmiana 1.

Zamienimy napis „To jest ETYKIETA” na „Wpisz swój login”.

Zmiana 2.

Usuniemy zdjęcie rysia, a w jego miejsce wstawimy napis: „Wpisz swoje hasło".

Zmiana 3.

Przycisk (Button) przesuniemy do miejsca, w którym wyświetlał się napis: „Kliknij Wyślij, aby potwierdzić swój wpis".

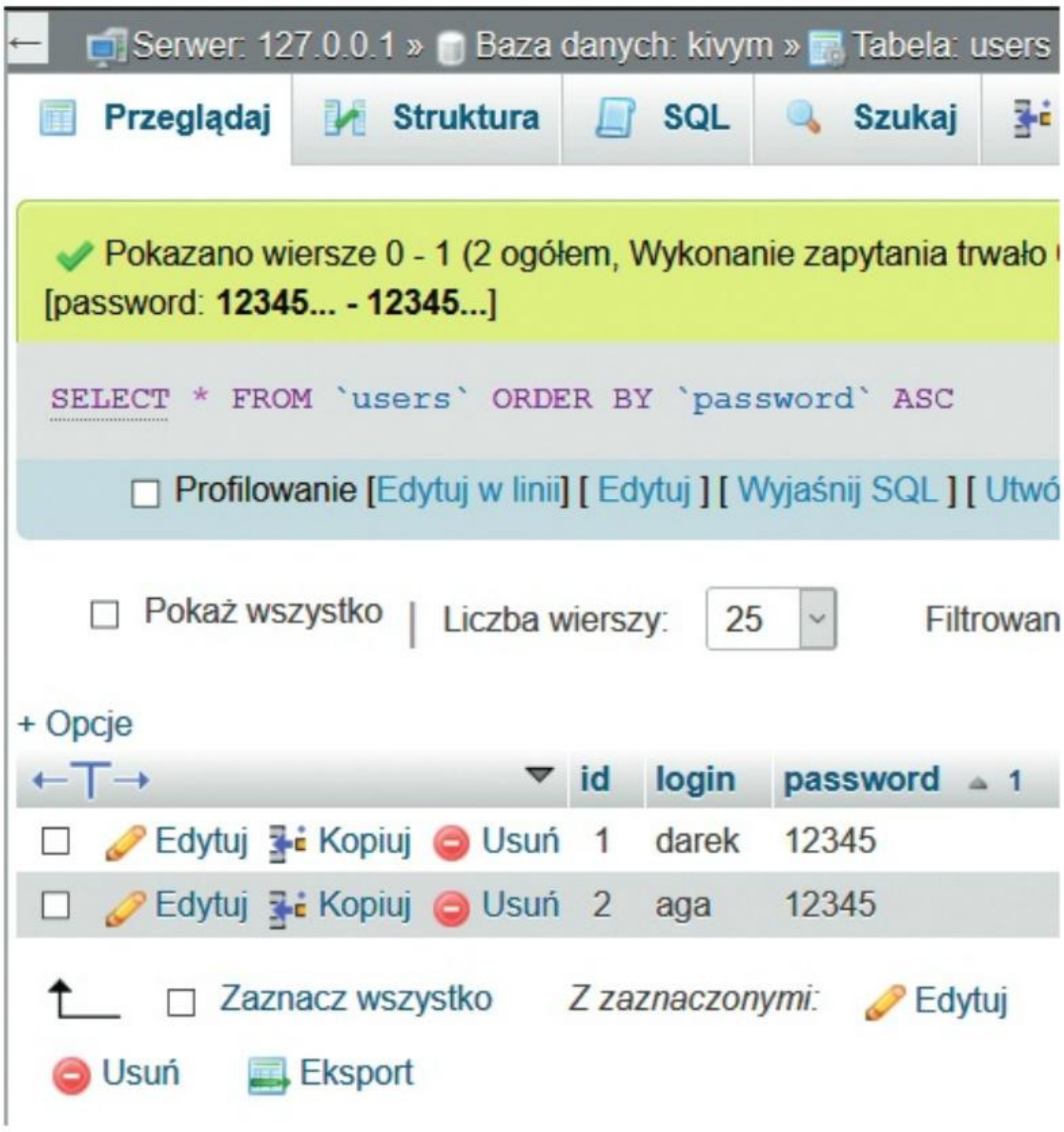
Zmiana 4.

W najniższej części prawej kolumny zamienimy tekst „Napisałaś / napisałeś:" na „Aby się zalogować: Wpisz login i hasło – kliknij przycisk Wyślij".

Zmiana 5.

Usuniemy linijkę zawierającą: **`from kivy.uix.image import Image`**. Nie będziemy używali zdjęć, więc nie jest nam potrzebna funkcjonalność dostarczana przez ten fragment kodu.

Najpierw zajmiemy się obsługą bazy danych MariaDB. Zbudujemy panel logowania do aplikacji. Będą potrzebne dane, których sprawdzenie umożliwi uzyskanie statusu użytkownika zalogowanego. Takie zasoby powinny się znaleźć w bazie MariaDB – jak na rys. 16.45.



Rys. 16.45. Widok danych niezbędnych do testowania Formularza logowania – phpMyAdmin

Możesz je dodać ręcznie do bazy danych za pomocą menedżera baz danych phpMyAdmin lub skorzystać z poniżej opisanego kodu. Aby stworzyć w bazie 'kivym' tabelę **users** z przykładowymi danymi, użyj kodu takiego jak rys. 16.46.

```
import mysql.connector

baza = mysql.connector.connect(
  host="localhost",
  user="root",
  password=""
)

poloczenie = baza.cursor()

createTableUsers = "CREATE TABLE kivym.users (id INT
AUTO_INCREMENT PRIMARY KEY, login VARCHAR(25),
password VARCHAR(50))"
insertInToUsers = "INSERT INTO kivym.users VALUES
(null, 'darek', '12345', (null, 'aga', '12345')"

poloczenie.execute(createTableUsers)
poloczenie.execute(insertInToUsers)

baza.commit()
```

Rys. 16.46. Kod wykonany w językach Python i SQL – dodanie do relacyjnej bazy danych MariaDB (przykładowe rekordy z danymi)

Teraz można rozpocząć kodowanie panelu, który:

- umożliwi wpisanie przez użytkownika loginu i hasła;
- pobierze wpisane dane;
- sprawdzi poprawność danych z zasobami w bazie danych;
- poinformuje użytkownika o poprawności logowania lub o błędnie wpisanych danych.

```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
import mysql.connector

# Stworzenie klasy apki Formularz, której układ strony
wykorzystuje element UI GridLayout
class Formularz(GridLayout):

    # Inicjalizacja aplikacji
    def __init__(self, **kwargs):
        super(Formularz, self).__init__(**kwargs)

        # Wstaw kolumny - określa liczbę kolumn w układzie
        GridLayout
        self.cols = 2
```

Rys. 16.47. Kod wykonany w języku Python – framework Kivy, aplikacja Formularz logowania (część 1)

W kolejnej części kodu znajdziesz nowy element:

```
self.login = TextInput(
hint_text='Wpisz login',size_hint_x=3, padding=[28, 24, 3, 1]
)
```

Elementy kodu zawarte wewnątrz nawiasu po TextInput powodują:

- **`hint_text`** – ukazanie się wyszarzonej podpowiedzi dla użytkownika w polu TextInput (w technologii webowej nazywaliśmy ten element placeholder);
- **`size_hint_x=3, padding=[28, 24, 3, 1]`** – wyrównanie w pionie i poziomie tekstu zarówno tego, który występuje w wyniku użycia **`hint_text`**, jak i docelowego – wpisywanego przez użytkownika (rys. 16.48–16.53).

W tym przypadku właściwość `size_hint_x=3` można nawet pominąć. Z kolei dzięki `padding=[28, 24, 3, 1]` można zapanować nad marginesami wewnętrznymi, które są ustawione w kolejności padding=[lewy, górny, prawy, dolny]. Pamiętaj, że ustawianie par naprzeciwległych, czyli lewy–prawy, górny–dolny, powoduje, iż mogą na siebie wzajemnie oddziaływać, powodując nieoczekiwane ustawienia tekstu.

```
# Dodaj widget z Label (etykietą)
self.label = self.add_widget(Label(text="Wpisz swój
login",font_size ="17"))

# Dodaj Pole Input (Box)
self.login = TextInput(hint_text='Wpisz login',
size_hint_x=3, padding=[28, 24, 3, 1])
self.login.background_color = [1, 1, 0, 1]
self.add_widget(self.login)

# Dodaj widget z etykietą (Label)
self.label2 = self.add_widget(Label(text="Wpisz swoje
hasło",font_size ="17"))

# Dodaj Pole Input (Box)
self.haslo = TextInput(hint_text='Wpisz hasło',
multiline=False,size_hint_x=3, padding=[28, 24, 3, 1])
self.haslo.background_color = [1, 1, 0, 1]
self.add_widget(self.haslo)
```

Rys. 16.48. Kod wykonany w języku Python – framework Kivy, aplikacja Formularz logowania (część 2)

```
# Przycisk interaktywny umożliwiający uruchomienie
zdarzenia w aplikacji (Submit Button)
self.submit = Button(text="Wyślij", font_size=22)
self.submit.background_color = [1, 0, 0, 1]
# Dodaj widget - przycisk do wysyłania (Submit Button)
self.add_widget(self.submit)
# Dodanie obsługi kliknięcia (czyli zdarzenia) do
przycisku - po kliknięciu zostanie wywołana funkcja o
nazwie wypiszTekst
self.submit.bind(on_press=self.sprawdzLoginWBazie)
```

Rys. 16.49. Kod wykonany w języku Python – framework Kivy, aplikacja Formularz logowania (część 3)

```
# Funkcja obsługująca kliknięcie w przycisk Wyślij -
sprawdzi,czy istnieją w bazie SQL - dane wpisane do inputów
login oraz hasło
def sprawdzLoginWBazie(self, instance):
    # przechwyć do nowej zmiennej tekst wpisany do Inputa
    login
    namePrzechwycone = self.login.text
    # przechwyć do nowej zmiennej tekst wpisany do Inputa
    haslo
    hasloPrzechwycone = self.haslo.text

    # Ustanowienie połączenia z bazą danych SQL o nazwie
    kivym
    baza = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="kivym"
    )
```

Rys. 16.50. Kod wykonany w języku Python – framework Kivy, aplikacja Formularz logowania (część 4)

```
# Przygotowanie zapytania do bazy SQL dotyczącego Loginu
zapytanieLogin = """SELECT login FROM users WHERE
login=%s"""
# Przygotowanie zapytania do bazy SQL dotyczącego Hasła
zapytanieHaslo = """SELECT password FROM users WHERE
login=%s"""

# Ustanowienie połączenia, wstawienie zapytania do query
oraz przechwycenie wyniku dla Loginu
poloczenie = baza.cursor()
poloczenie.execute(zapytanieLogin, (namePrzechwycone, ))
wynikLogin = poloczenie.fetchall()

# Ustanowienie połączenia, wstawienie zapytania do query
oraz przechwycenie wyniku dla Hasła
poloczenie = baza.cursor()
poloczenie.execute(zapytanieHaslo, (namePrzechwycone, ))
wynikHaslo = poloczenie.fetchall()
```

Rys. 16.51. Kod wykonany w języku Python – framework Kivy, aplikacja Formularz logowania (część 5)

```
# Konstrukcja warunkowa:
# if sprawdzi,czy login wpisany do pola input istnieje
if len(wynikLogin) > 0 and wynikHaslo[0][0] ==
hasloPrzechwycone:

    # Stworzenie widgetu informującego o aktywnym
    zalogowaniu
    self.add_widget(Label(text=f'Jesteś zalogowany jako:
    {namePrzechwycone} '))

    # Funkcjonalność powodująca wyczyszczenie pola Input
    po przechwyceniu wiadomości
    self.login.text = ""
    self.haslo.text = ""
```

Rys. 16.52. Kod wykonany w języku Python – framework Kivy, aplikacja Formularz logowania (część 6)

```
    # else obsłuży aplikację, gdy login wpisany do
    pola input nie istnieje lub podane hasło nie
    zgadza się z tym, które dla danego loginu jest
    przechowywane w bazie danych
    else:
        self.add_widget(Label(text=f'Podałeś/aś zły
        login'))

        # Funkcjonalność powodująca wyczyszczenie
        pola Input po przechwyceniu wiadomości
        self.login.text = ""
        self.haslo.text = ""

class MobileApp(App):
    def build(self):
        return Formularz()

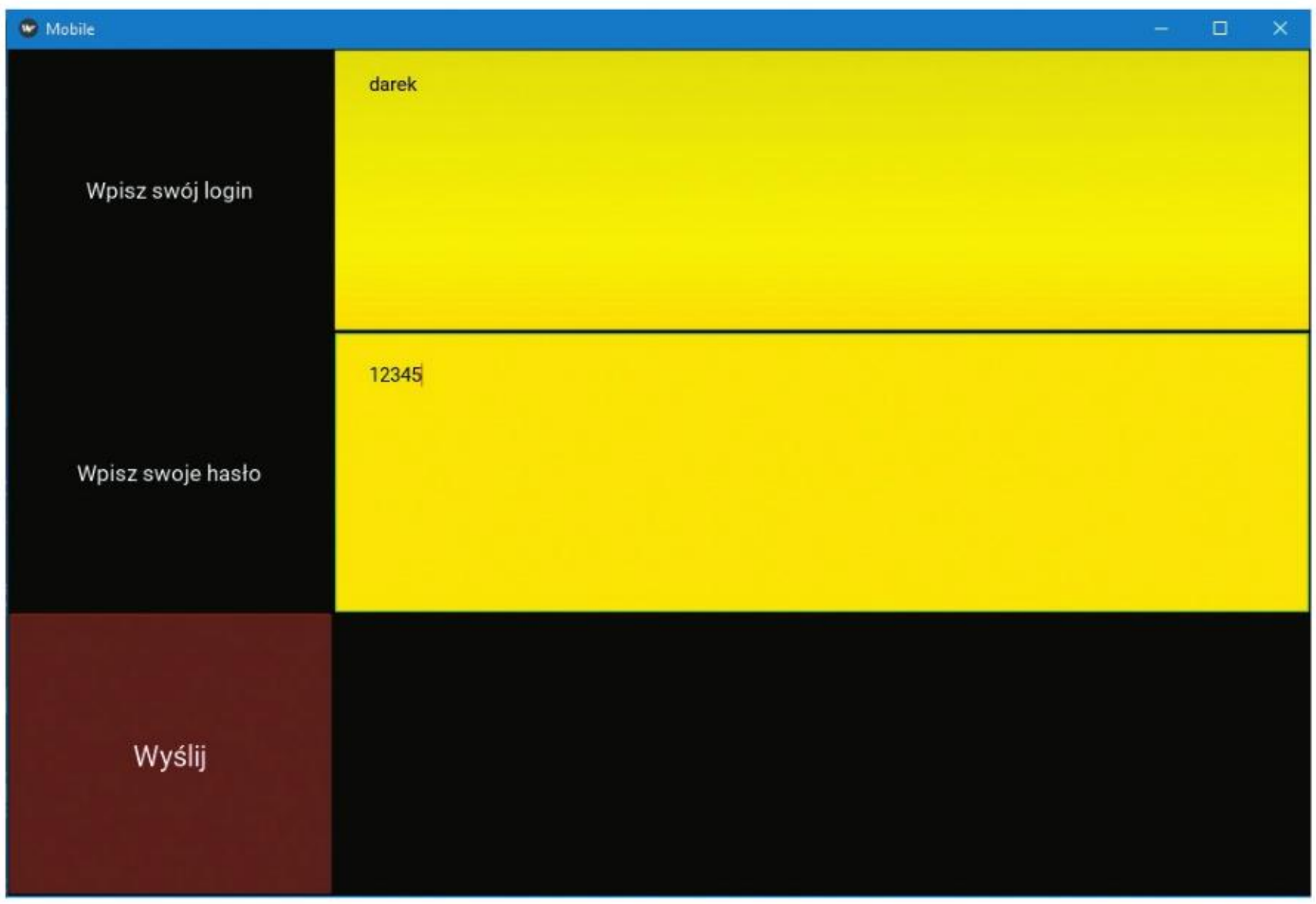
if __name__ == '__main__':
    MobileApp().run()
```

Rys. 16.53. Kod wykonany w języku Python – framework Kivy, aplikacja Formularz logowania (część 7)

Na rys. 16.54–16.57 przedstawiono wygląd panelu logowania.



Rys. 16.54. Aplikacja Formularz logowania – widok początkowy



Rys. 16.55. Aplikacja Formularz logowania – widok wpisania poprawnych loginu i hasła



Rys. 16.56. Aplikacja Formularz logowania – widok po wpisaniu poprawnych loginu i hasła (po potwierdzeniu zgodności danych z bazy na dole pojawiła się informacja: „Jesteś zalogowany jako: darek")



Rys. 16.57. Aplikacja Formularz logowania – widok po wpisaniu niepoprawnego loginu i/lub hasła (po potwierdzeniu niezgodności danych z bazy na dole pojawiła się informacja: „Podałaś / podałeś zły login")

Program informuje tylko o tym, że jest błędnie wpisany login, tzn. zalogowanie nastąpi tylko wtedy, kiedy zostaną wpisane poprawnie login i hasło. Ta część działa poprawnie. Brakuje jednak informacji o błędnie wpisanym haśle. Uzupełnienie kodu będzie twoim zadaniem na koniec rozdziału.

16.8. Ekrany aplikacji mobilnej

Aplikacja mobilna kojarzy się zwykle z kilkom ekranami, pomiędzy którymi jest zbudowana nawigacja oparta na kliknięciach lub na przesuwaniu ekranu palcem. Zbudujemy taki element UI, który będzie się składał z dwóch ekranów.

Rozpoczynamy od stworzenia pliku, np. multipleWin.py. Poniżej znajdziesz kod, który trzeba wpisać do wspomnianego pliku. Na rys. 16.58 znajdziesz u góry pliku import przykładowych modułów Kivy, dających nam pożądaną funkcjonalność. Kolejne linijki:

```
class FirstWindow(Screen):
    pass
class SecondWindow(Screen):
    pass
class WindowManager(ScreenManager):
    pass
```

utworzą dwa ekrany i menedżera, który będzie nimi zarządzał.

Ostatnia linijka umożliwia połączenie pliku Pythona z plikiem kv:

```
kv = Builder.load_file('multipleWin.kv')
```

```
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.properties import ObjectProperty
from kivy.lang import Builder
from kivy.uix.screenmanager import ScreenManager,
Screen


#Zdefiniuj pierwszy ekran
class FirstWindow(Screen):
    pass

#Zdefiniuj drugi ekran
class SecondWindow(Screen):
    pass
```

Rys. 16.58. Plik multipleWin.py – początek kodu, który służy zbudowaniu aplikacji mobilnej składającej się z dwóch ekranów

```
class WindowManager(ScreenManager):
    pass

kv = Builder.load_file('multipleWin.kv')
```

Rys. 16.58. Cd. Plik multipleWin.py – początek kodu, który służy zbudowaniu aplikacji mobilnej składającej się z dwóch ekranów

Na rys. 16.59 znajdziesz elementy, które są dopełnieniem kodu Pythona, zapewniającym funkcjonalność tworzonej aplikacji.

```
class MojLayout(Widget):
    pass

class FantasticApp(App):
    def build(self):
        return kv

if __name__ == '__main__':
    FantasticApp().run()
```

Rys. 16.59. Plik multipleWin.py – druga część kodu, który służy zbudowaniu aplikacji mobilnej składającej się z dwóch ekranów

Plik multipleWin.py miał prostą budowę. Teraz zajmiemy się kolejnym plikiem, który nazwiemy np. multipleWin.kv. Będzie on oparty na języku frameworka Kivy – kv. Za pomocą tego języka zbudujemy elementy interfejsu aplikacji mobilnej, a konkretnie dwa ekrany. Projekt będzie miał prostą strukturę i bez problemu będziesz mógł rozbudować aplikację o dowolną liczbę ekranów. Wystarczy, że wprowadzisz adekwatne zmiany w kodach plików multipleWin.py i multipleWin.kv.

Na początku pliku wskazujemy, w jakiej wersji frameworka Kivy pracujemy:

```
#:kivy 2.0.0.
```

Bez wpisania powyższej linijki kodu aplikacja będzie działać, jednak walidatory kodu działające w niektórych IDE, np. Visual Studio Code, mogą wskazywać błędy.

Następnie zaimportujemy funkcjonalność Kivy związaną z użyciem systemu szesnastkowego podczas określania kolorów:

```
#:import utils kivy.utils.
```

Funkcjonalność pochodząca z powyższej linijki kodu zostanie wykorzystana w jednej z kolejnych sekcji:

```
<SecondWindow>, a konkretnie w linii:
background_color: utils.get_color_from_hex('#ED01DE')
```

Ostatnie trzy linijki kodu tworzą mechanizm zarządzający WindowManager, a wewnątrz niego dwa elementy, które będą stanowiły poszczególne ekrany aplikacji:

```
FirstWindow:
SecondWindow:
```

```
#:kivy 2.0.0
#:import utils kivy.utils

WindowManager:
    FirstWindow:
    SecondWindow:
```

Rys. 16.60. Plik multipleWin.kv – pierwsza część kodu, który służy do zbudowania interfejsu aplikacji mobilnej składającej się z dwóch ekranów

Na rys. 16.61 i 16.62 przedstawiono kody tworzące interfejs odpowiednio dla pierwszego i drugiego ekranu aplikacji. Po deklaracji elementu:

```
FirstWindow
```

przypisujemy dla niego nazwę:

```
name: "pierwszy",
```

która umożliwi nam wskazanie tego elementu w nawigacji. Zastosowanie nadanej nazwy znajdziesz nie na kolejnym, ale na następującym po nim zdjęciu w sekcji:

```
on_release:
            app.root.current = "pierwszy"
    root.manager.transition.direction = "up"
```

Dzięki poleceniu **app.root.current** płynnie przechodzi się do ekranu o nazwie „pierwszy". W sekcji:

```
BoxLayout:
```

znajdziesz elementy UI aplikacji, czyli:

```
Label:
Button:
```

Zwróć uwagę na kod:

```
on_release:
            app.root.current = "drugi"
    root.manager.transition.direction = "left"
```

Jeżeli naciśniesz element Button, w którym powyższy kod jest zawarty, wówczas nastąpi animacja – płynne przejście do ekranu zdefiniowanego pod nazwą „drugi".

W przykładzie zastosowano animację „left", czyli element przesunie się w lewo. Możliwe są inne ustawienia:

- **"right"**,
- **"up"** – zastosowane w przejściu z **<SecondWindow>** do **<FirstWindow>**,
- **"down"**.

Dalej w kodzie ekranu **<FirstWindow>** znajdziesz:

```
color: (0,1,0,1)
    background_color: (238/250,50/250,0/250,1)
```

Powyższy zapis wskazuje na standardowe dla frameworka Kivy i języka kv określanie kolorów, czyli RGBA. Części ułamkowe poszczególnych elementów w nawiasie dają określone kolory.

Alternatywne rozwiązanie kwestii koloru omówiono już w związku z importem utils kivy.utils.

```
<FirstWindow>:
    name: "pierwszy"

    BoxLayout:
        orientation: "horizontal"
        size: root.width, root.height

        Label:
            text: "Pierwszy Ekran"
            font_size:32

        Button:
            text: "Next >>"
            font_size:32
            on_release:
                app.root.current = "drugi"
                root.manager.transition.direction =
                "left"
            color: (0,1,0,1)
            background_color: (238/250,50/250,0/250,1)
```

Rys. 16.61. Plik multipleWin.kv – druga część kodu, który służy do zbudowania interfejsu aplikacji mobilnej składającej się z dwóch ekranów

Doszliśmy do ostatniego elementu kodu – spójrz na rys. 16.62. Buduje on drugi ekran **<SecondWindow>**. Zawiera niemal takie same elementy jak ten pierwszy **<FirstWindow>**. Różnica jest tylko w:

- elemencie, do którego jest dokonywane przekierowanie, i w sposobie wykonania animacji – będzie to ruch w górę:

```
on_release:
    app.root.current = "pierwszy"
    root.manager.transition.direction = "up"
```

- sposobie określenia koloru tła – dzięki zaimportowaniu funkcjonalności z utils kivy.utils korzystamy z systemu szesnastkowego:

```
background_color: utils.get_color_from_hex('#ED01DE')
```

```
<SecondWindow>:
    name: "drugi"

    BoxLayout:
        orientation: "vertical"
        size: root.width, root.height

        Label:
            text: "Drugi Ekran"
            font_size:32

        Button:
            text: "<< Back"
            font_size:32
            on_release:
                app.root.current = "pierwszy"
                root.manager.transition.direction =
                "up"
            color: (155/255.0,250/250,0/255.0,1)
            background_color: utils.get_color_from_hex
            ('#ED01DE')
```

Rys. 16.62. Plik multipleWin.kv – trzecia część kodu, który służy do zbudowania interfejsu aplikacji mobilnej składającej się z dwóch ekranów

Budowa dwóch ekranów aplikacji mobilnej jest we frameworku Kivy całkiem prosta. Na rys. 16.63 i 16.64 przedstawiono wywołanie aplikacji w systemie operacyjnym Windows 10. Aby tego dokonać:

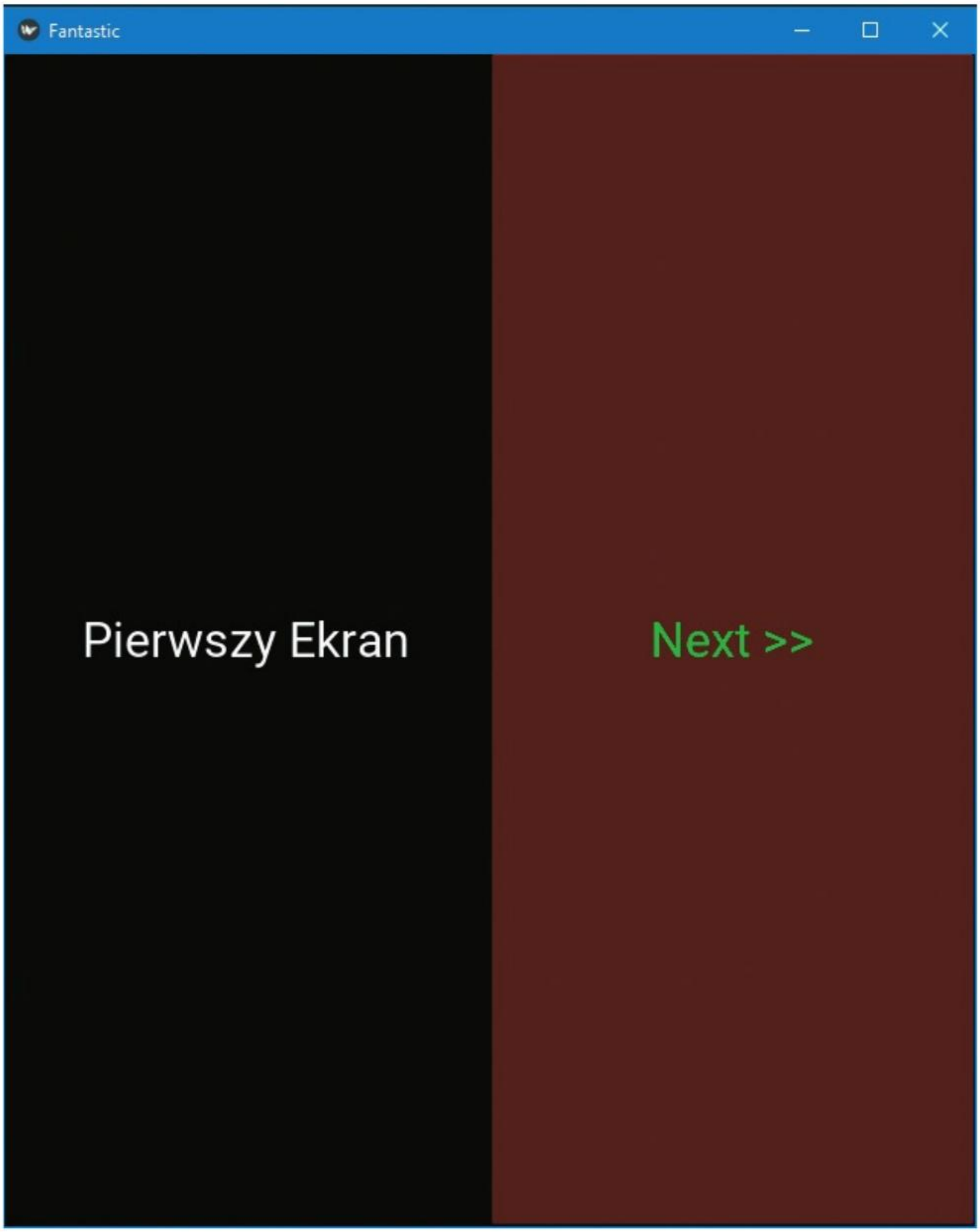
- otwórz konsolę systemową (lub inną, za której pomocą można otworzyć plik Pythona);
- przejdź poleceniem cd do katalogu stworzonego projektu;
- uruchom kod aplikacji poleceniem właściwym dla twojego systemu, np.:

```
python multipleWin.py
```

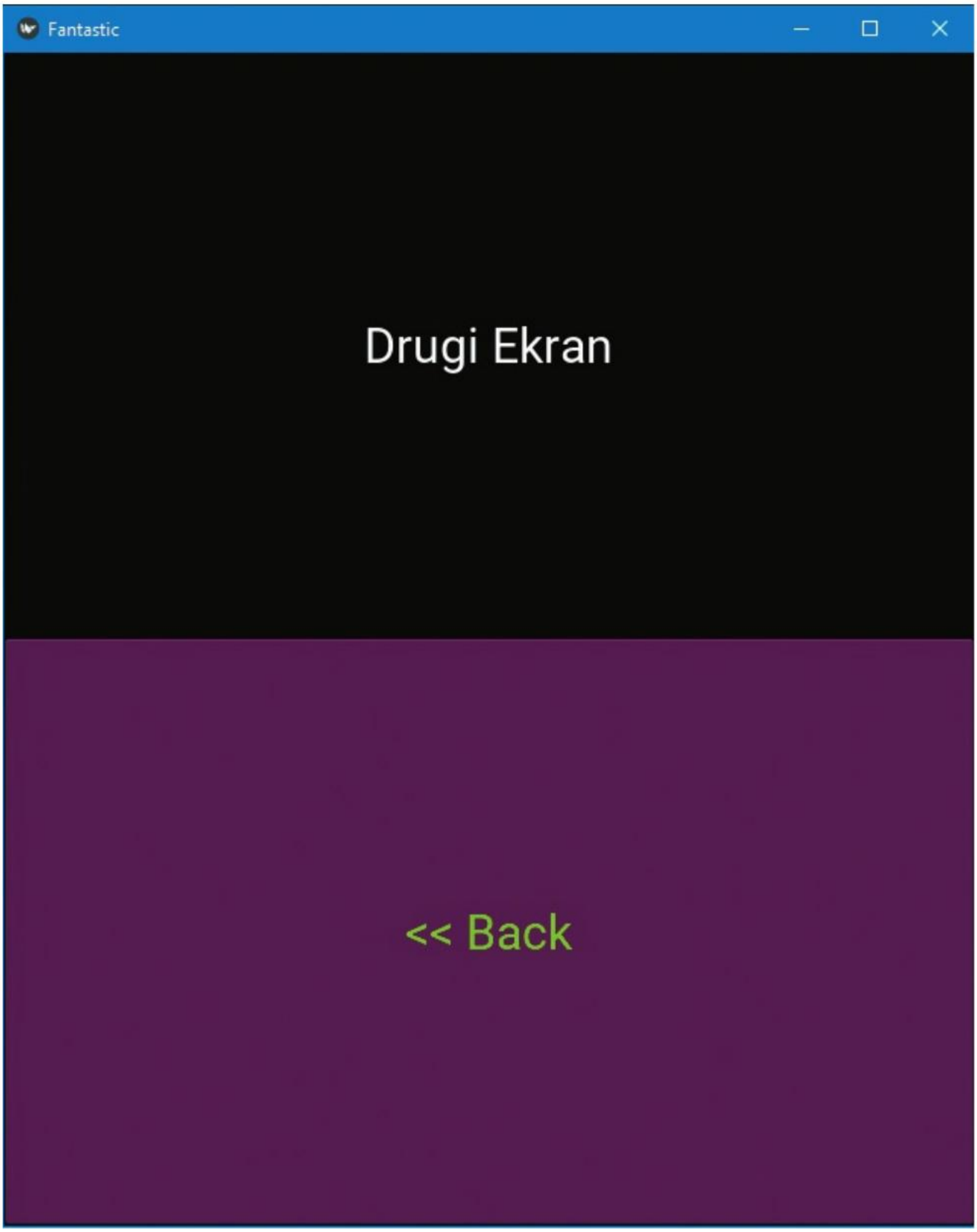
CIEKAWOSTKA

Bardzo dobrym przyzwyczajeniem jest uruchamianie plików stworzonych w języku programowania Python poleceniem `python`. Jest ono właściwie rozumiane we wszystkich środowiskach. Istnieje też jego krótsza odmiana – `py`, czyli kod zapisany powyżej możesz także wywołać za pomocą:

```
py multipleWin.py
```



Rys. 16.63. Widok aplikacji mobilnej – pierwszy ekran, który powstał w wyniku uruchomienia kodu frameworka Kivy, pliki multipleWin.py i multipleWin.kv



Rys. 16.64. Widok aplikacji mobilnej – drugi ekran, który powstał w wyniku uruchomienia kodu frameworka Kivy, pliki multipleWin.py i multipleWin.kv

16.9. Omówienie przygotowania aplikacji do publikacji w sklepie – podsumowanie

Aby podzielić się z innymi osobami stworzoną aplikacją (bezpłatnie lub odpłatnie), trzeba zamienić kod Kivy/Python na taki zestaw (paczkę), który zadziała na wybranym urządzeniu i w jego systemie operacyjnym. Przypomnijmy kolejne kroki, które trzeba wykonać, aby osiągnąć produkt nadający się do użytkowania w systemie operacyjnym Android.

1. **Wdrażanie aplikacji KivyPython na Androida za pomocą aplikacji Buildozer**
   Wdrażanie aplikacji w języku Python i Kivy na telefon z systemem Android za pomocą komputera z systemem Windows, na którym musisz mieć jedno z dwóch rozwiązań:
   - Linux (np. Ubuntu 18 lub 20) w VirtualBox;
   - Linux (np. Ubuntu), pobrany ze sklepu Microsoft Store i zainstalowany w ramach WSL2 (Buildozer nie działa bezpośrednio w systemie Windows).

   Alternatywą jest dysponowanie systemem operacyjnym Linux (np. Ubuntu[3]) na nawet nieco starszym komputerze.
2. **Podłącz telefon z systemem Android do komputera i upewnij się, że jest włączona opcja zezwalająca na debugowanie USB**
   Istnieje wiele wersji oprogramowania na urządzenia mobilne. Zazwyczaj trzeba najpierw aktywować opcje deweloperskie.
3. **Buildowanie – Buildozer**
   Gdy masz już przygotowane urządzenie mobilne, możesz zacząć pracę z buildowaniem. W trakcie pracy aplikacji Buildozer telefon może poprosić o akceptację dla debugowania USB. Nie możesz przegapić tego momentu, gdyż brak akceptacji może oznaczać fiasko dalszego działania wspomnianego programu.

Przejdźmy do systemu Linux. Otwórz terminal (na maszynie wirtualnej lub zwykłej instalacji Linuksa) i sprawdź posiadane (działające) wersje Pythona i jego menedżera pakietów – programu pip. W tym celu uruchom polecenia przedstawione na rys. 16.65.

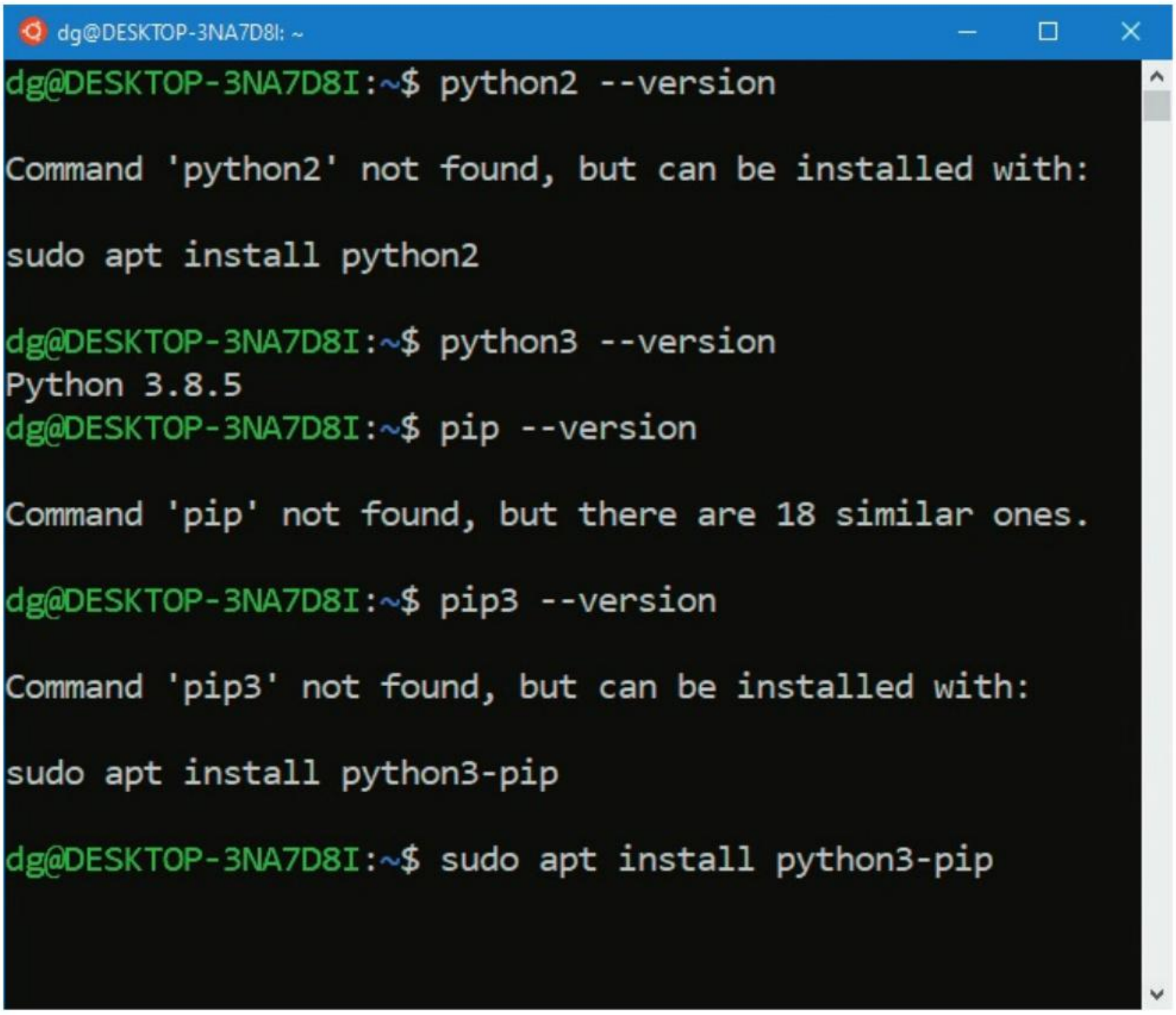
Sprawdź, która wersja Pythona jest zainstalowana na systemie operacyjnym Linux. Możesz użyć polecenia:

```
python -version
python3 --version
sudo apt-get install python3.9
```

Jeżeli jest zainstalowany python2.x, to być może trzeba będzie go odinstalować poleceniem:

```
sudo apt purge -y python2.7-minimal
```

[3] Pełną wersję instalacyjną Ubuntu możesz znaleźć na oficjalnej stronie projektu: https://ubuntu.com/#download.



Rys. 16.65. Sprawdzenie instalacji programów Python i pip (menedżer pakietów)

Instalacja Buildozera

Krok 1

Wykorzystaj repozytorium GIT:

```
sudo apt install git
git clone https://github.com/kivy/buildozer.git
cd buildozer
sudo python setup.py install
```

Za pomocą polecenia pip:

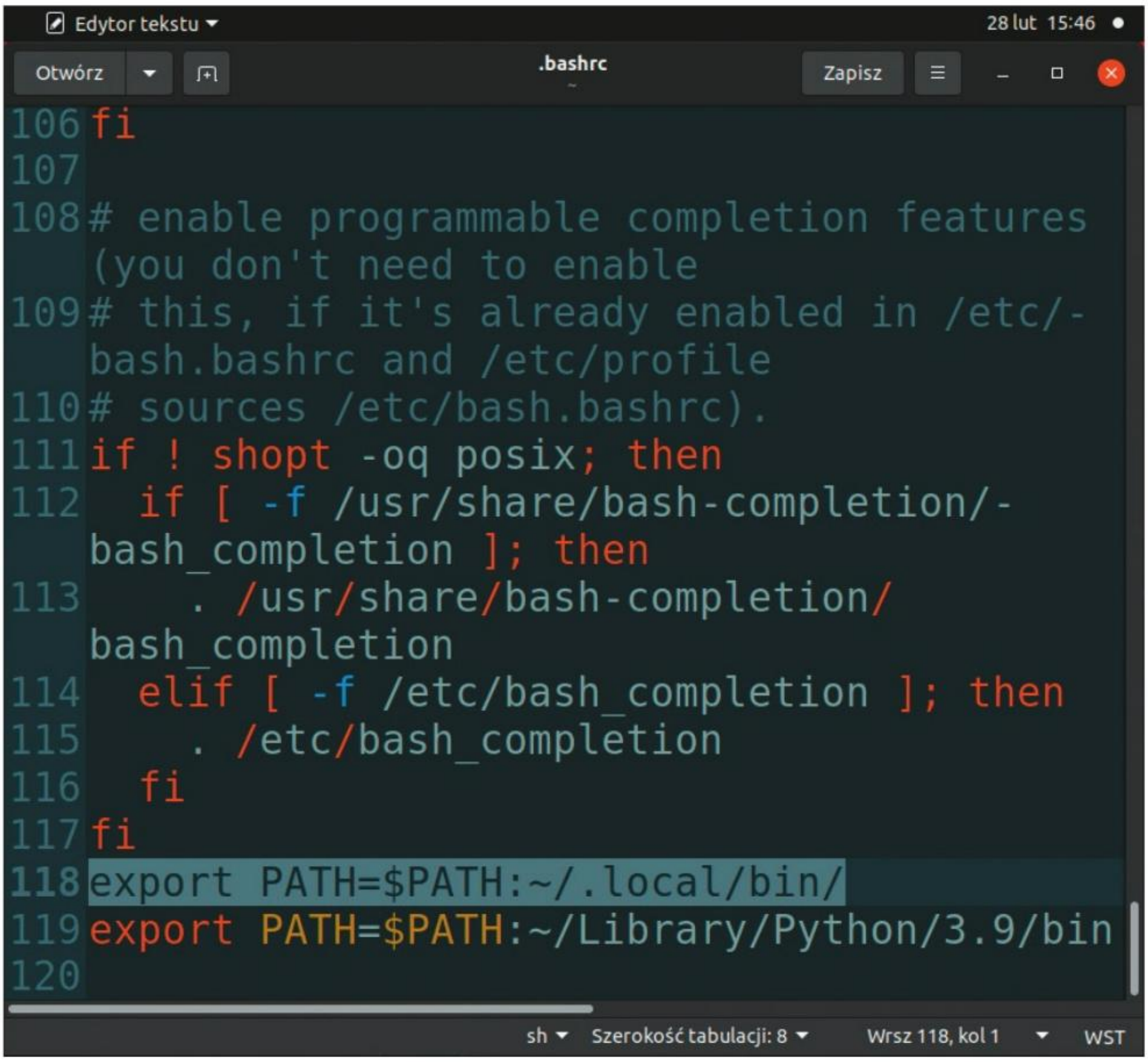
```
pip3 install --user --upgrade buildozer
```

Polecenia najlepiej wykonywać w ramach wirtualnego środowiska Pythona. Zwykle sprawia on mniej kłopotów z wykonaniem procesu buildowania. Jest to zalecana metoda.

Krok 2

Dodaj do pliku ~/.bashrc na samym końcu:

```
export PATH=$PATH:~/.local/bin/
restart Terminala!
```



Rys. 16.66. Dodanie do pliku .bashrc linijki kodu: export PATH=$PATH:~/.local/bin/

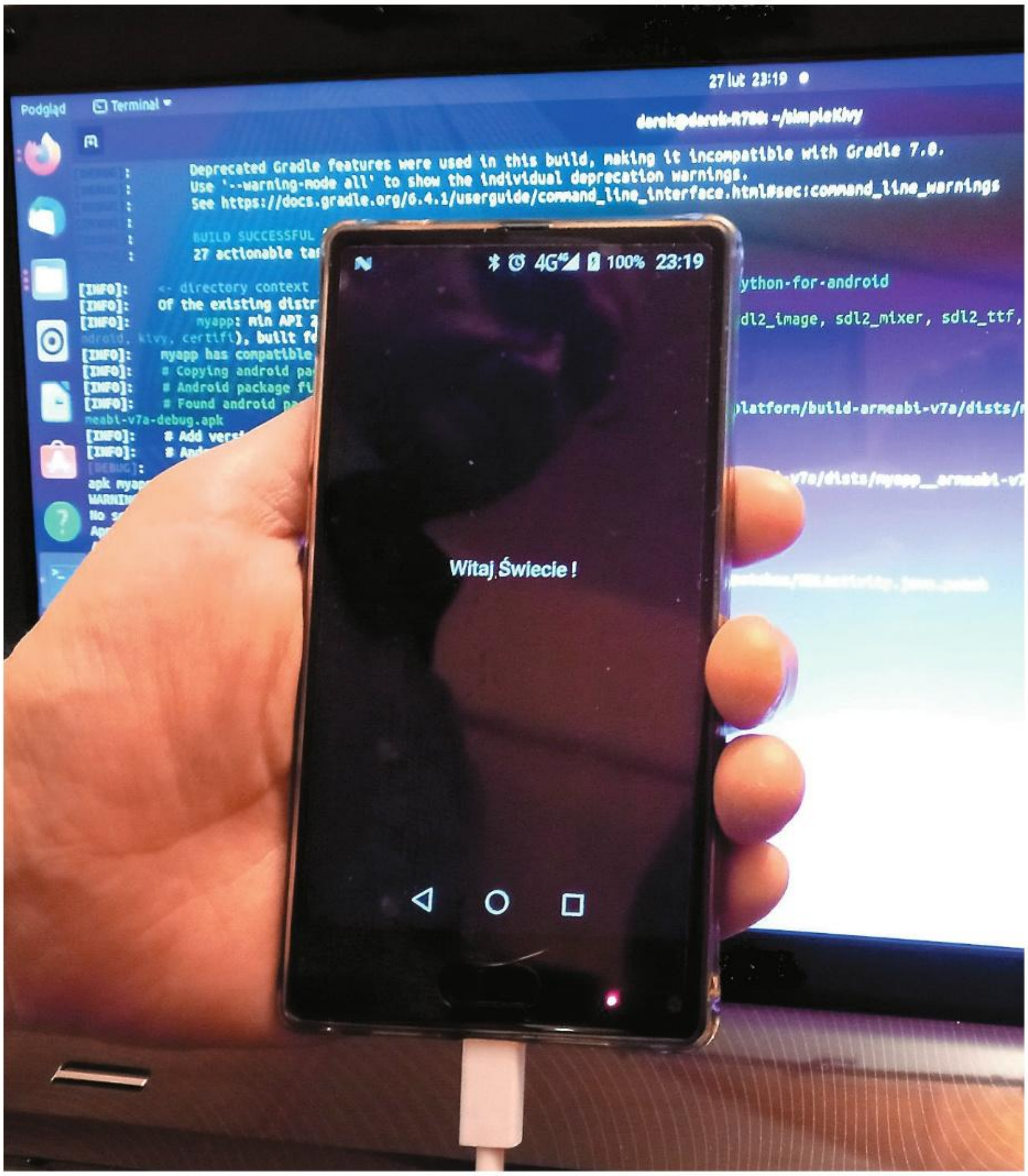
Krok 3

Wpisz kod:

```
sudo apt-get install -y python3-setuptools
cd simpleKivy      // przejście do katalogu stworzonej aplikacji Kivy;
                   katalog może wymagać odblokowania w Linuksie poleceniem:
                   chmod u=rwx,g=rwx,o=rwx simpleKivy
buildozer init     // stworzy plik z ustawieniami buildozer.spec
sudo apt update
```

```
sudo apt upgrade
sudo apt install -y git zip unzip openjdk-8-jdk python3-pip au-
toconf libtool pkg-config zlib1g-dev libncurses5-dev libncur-
sesw5-dev libtinfo5 cmake libffi-dev
pip3 install --user --upgrade cython virtualenv
sudo apt-get install cython
buildozer android debug deploy run
```

Efektem końcowym działania Buildozera będzie uruchomienie na smartfonie z systemem Android aplikacji wykonanej w Kivy.



Rys. 16.67. Efekt końcowy działania Buildozera – widok na urządzeniu mobilnym aplikacji wykonanej w Kivy

Bądź na czasie!

Od czasu pisania powyższej instrukcji do dnia, kiedy ją czytasz, mogło się wiele zmienić w świecie programowania. Sugerujemy, by nie opierać się wyłącznie na przedstawionych wytycznych. Musisz je skorygować o najnowsze informacje związane z tworzeniem paczki na Androida (a także na inne systemy). Stosowne informacje znajdziesz na stronach opisywanych projektów, np.:

- https://kivy.org/doc/stable/guide/packaging-android.html#packaging-your-application-into-apk;
- https://github.com/kivy/buildozer;
  https://python-for-android.readthedocs.io/en/latest/quickstart/.

Problemy z wykonaniem wersji aplikacji na urządzenie mobilne

Podczas wskazanych działań możesz spotkać się z wieloma problemami. Zwykle będą one związane z jakimś elementem środowiska twojego systemu operacyjnego. Musisz wtedy dokładnie przeczytać w konsoli, jakiego rodzaju problem uniemożliwia ukończenie procesu buildowania. Następnie trzeba będzie wprowadzić korektę (np. w zależnościach – zarówno tych w systemie, jak i w środowisku wirtualnym).

Po wykonaniu polecenia: **`Buildozer android debug deploy run`** – na urządzeniu przenośnym pojawi się aplikacja gotowa do uruchomienia. Niestety, często próba uruchomienia aplikacji Kivy w Androidzie kończy się niepowodzeniem. Może to być spowodowane błędem powstałym w trakcie procesu buildowania. Zwykle są to jednak zabezpieczenia wspomnianego systemu operacyjnego. Możesz spróbować wprowadzić kilka zmian, np. w swoim środowisku, które usunie problem.

Przykładowo: zamiast polecenia:

```
buildozer android debug deploy run
```

użyj wersji:

```
buildozer -v android debug.
```

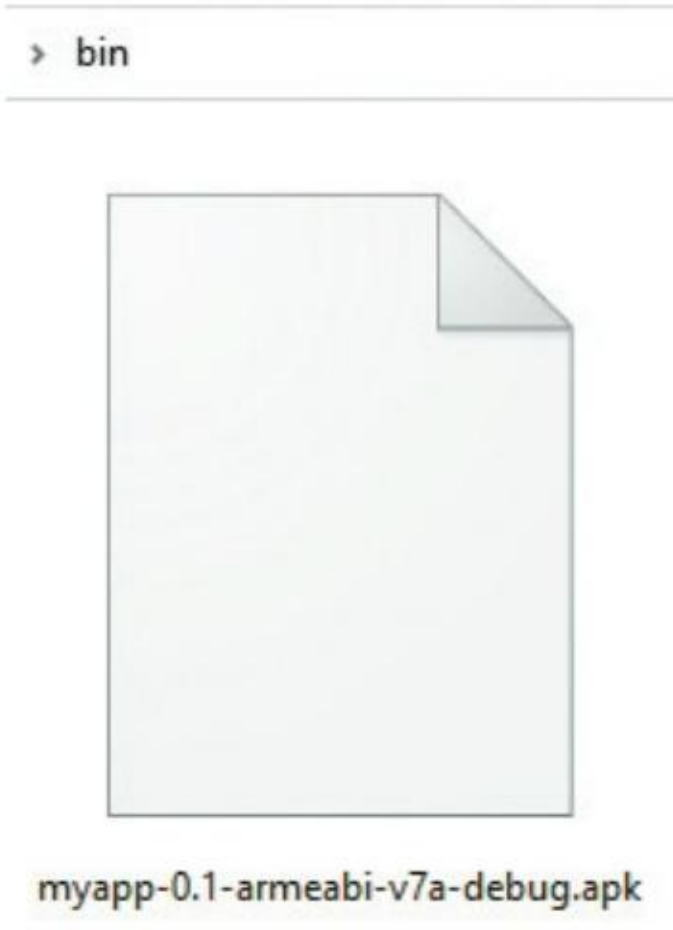
Efektem tego będzie stworzenie pliku .apk – jak na rys. 16.68.

CIEKAWOSTKA

Zgodnie z zaleceniami twórców frameworka Kivy w podręczniku przedstawiono wykonanie pliku .apk z wykorzystaniem programu Buildozer. Opisywane rozwiązanie testowano na trzech technologiach:

1. Linux Ubuntu 20.04 LTS, zainstalowany bezpośrednio na laptopie jako główny system operacyjny;
2. Linux Ubuntu 20.04 LTS, zainstalowany w programie VirtualBox;
3. Linux Ubuntu 20.04 LTS, zainstalowany w środowisku WSL na Windows.

Pomimo informacji znalezionych na wielu forach internetowych, że każdy z tych sposobów jest równie dobry do uzyskania poprawnego pliku .apk z projektu frameworka Kivy, zdecydowanie najłatwiej pracuje się z opcją numer 1.

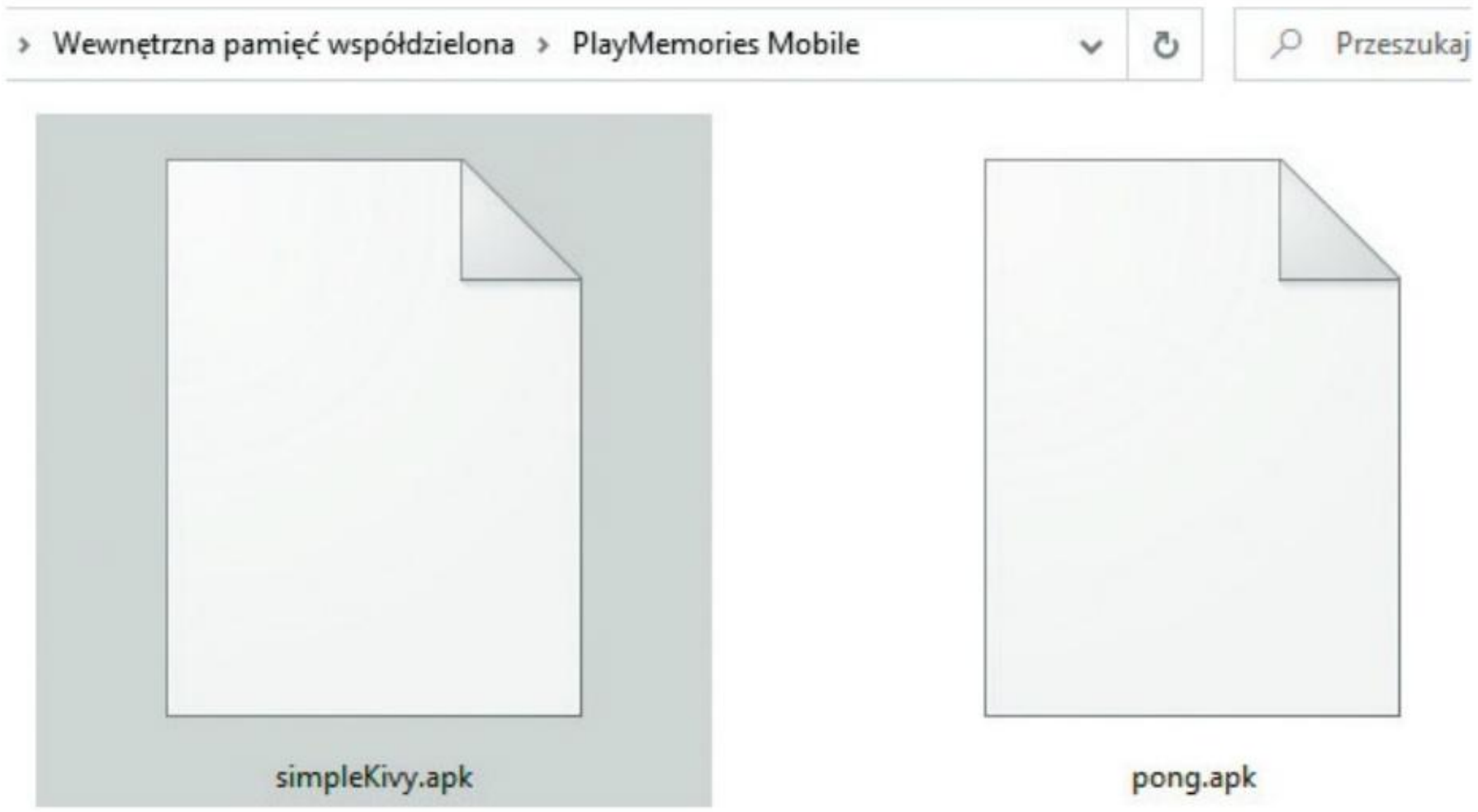


Rys. 16.68. Efekt końcowy działania aplikacji Buildozer – plik .apk

Teraz możesz zmienić nazwę wskazanego pliku na taką, jaka ma się wyświetlać na urządzeniu przenośnym. Pamiętaj, aby nie zmienić rozszerzenia – musi pozostać .apk (zmiana może spowodować uszkodzenie pliku).

Przekopiowanie pliku z komputera do urządzenia systemu mobilnego

Kolejnym krokiem musi być przekopiowanie pliku projektu mobilnego .apk z folderu w systemie operacyjnym Linux do katalogu w systemie Android. W przykładzie w tym podręczniku będzie to „Wewnętrzna pamięć współdzielona\PlayMemories Mobile". Jeżeli wystąpią problemy w komunikacji między Linuksem a Androidem, możesz przekopiować plik o rozszerzeniu .apk do katalogu Windowsa, a z niego przenieść do właściwego katalogu systemu mobilnego. Tego typu problem może się pojawić, gdy korzystasz z Linuksa zainstalowanego na VirtualBox, a także działającego w ramach technologii WSL2.

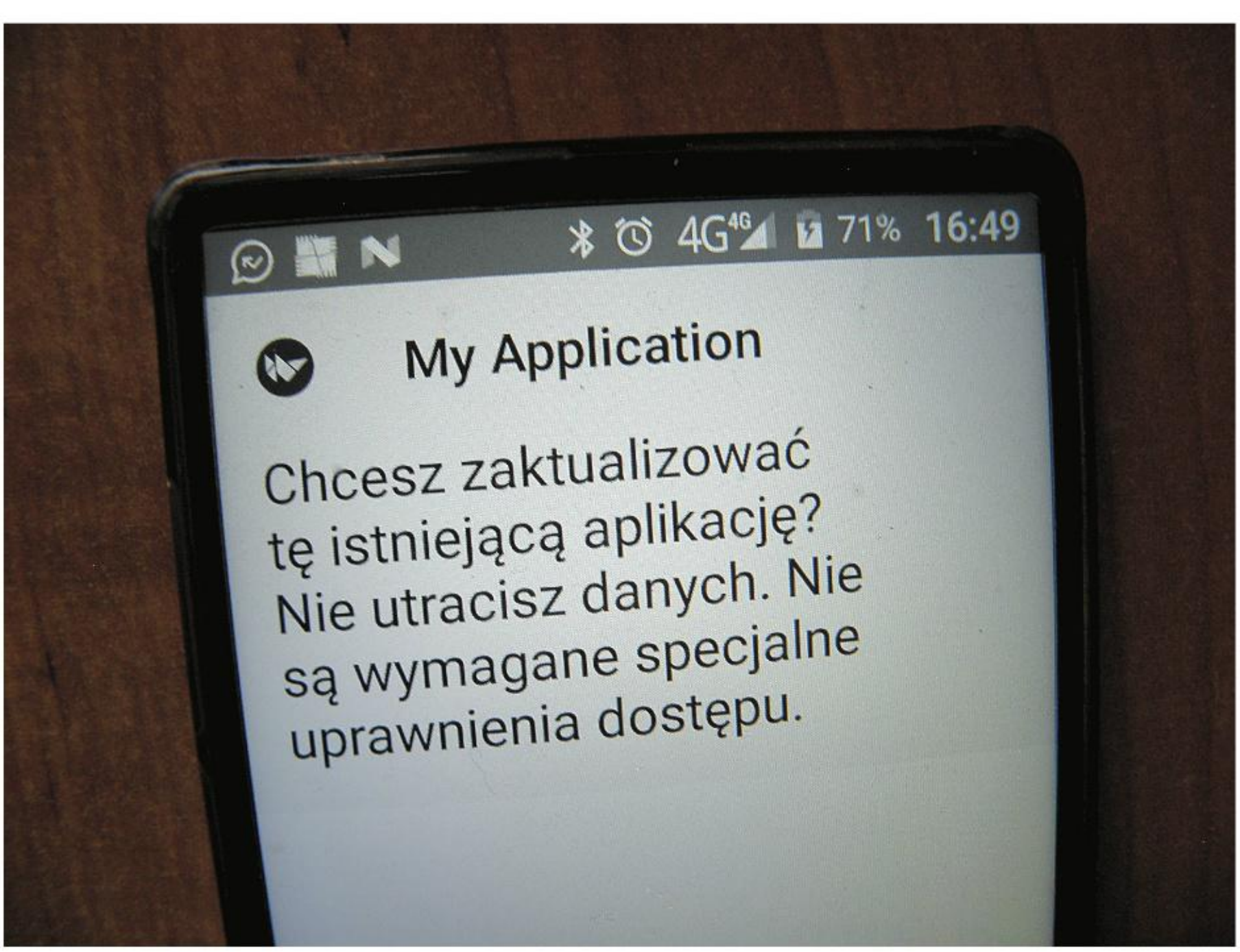


Rys. 16.69. Widok plików .apk w urządzeniu mobilnym z systemem Android

Urządzenie GSM systemu mobilnego Android

Aby zainstalować stworzoną aplikację na urządzeniu mobilnym, musisz w nią kliknąć. Prawdopodobnie wyskoczy ostrzeżenie ze strony systemu operacyjnego. Dopiero po potwierdzeniu chęci instalacji aplikacja zostanie zainstalowana. Na ekranie może się pojawić pytanie takie jak na rys. 16.70. Mnogość rozwiązań związanych z systemem Android i liczba producentów smartfonów bardzo komplikują dokonanie jednolitego opisu, który zakończyłby się sukcesem dla każdego. Zależy to od:

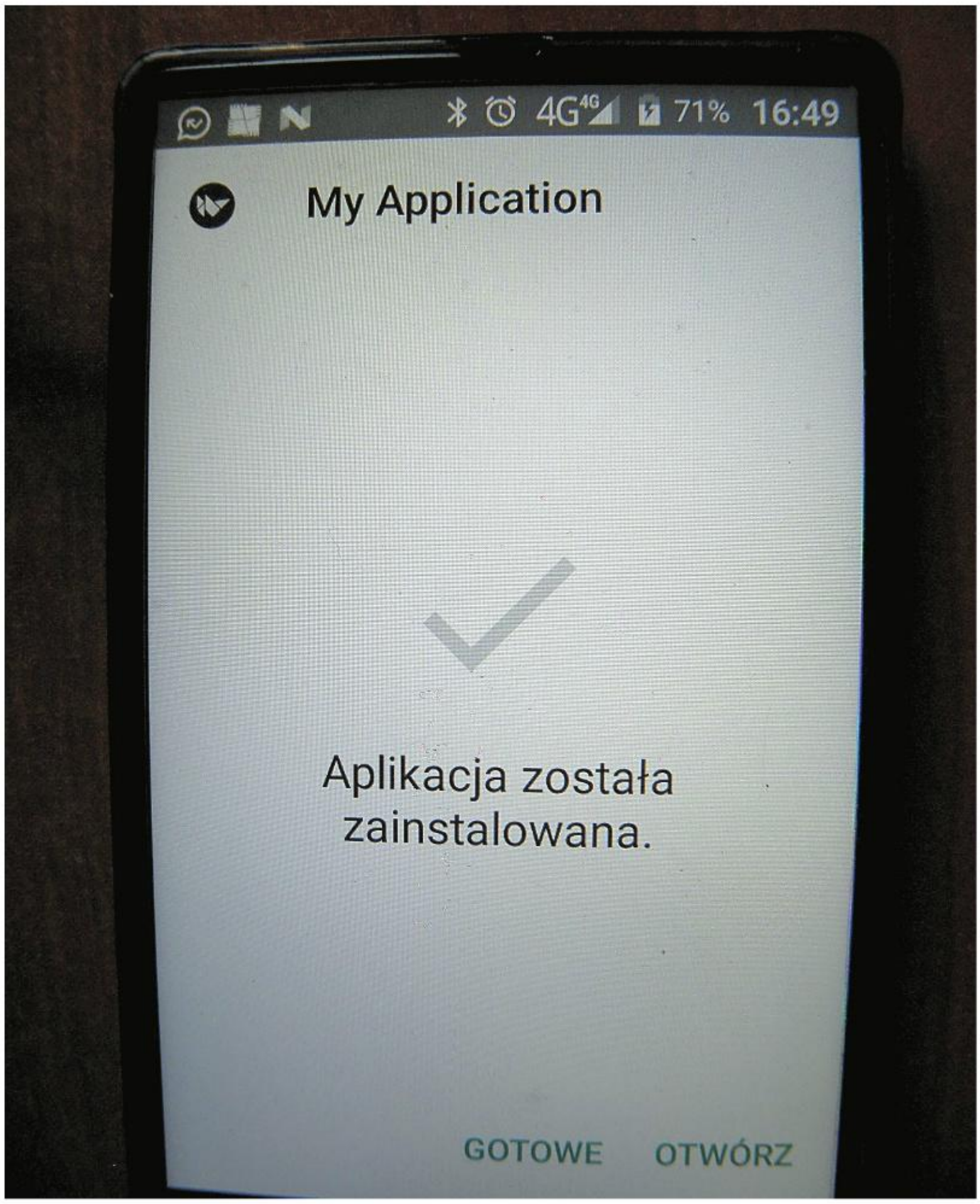
- wersji systemu Android;
- wersji urządzenia (producenta);
- poziomu zabezpieczeń, który jest ustawiony w danym systemie operacyjnym bądź w samym urządzeniu.



Rys. 16.70. Widok instalacji pliku .apk w urządzeniu mobilnym z systemem Android

Otworzenie aplikacji na urządzeniu mobilnym

Gdy zainstalujesz aplikację mobilną na swoim urządzeniu, najprawdopodobniej pojawi się ona wśród gier i/lub programów smartfona. Wystarczy tylko ją wyszukać i można zacząć jej używać.



Rys. 16.71. Widok potwierdzenia zainstalowania pliku .apk w urządzeniu mobilnym z systemem Android

Opis przygotowania do publikacji

Gdy stworzysz aplikację, możesz ją udostępnić publicznie. Istnieje kilka miejsc, z których potencjalni użytkownicy mogą pobrać i zainstalować twoją aplikację. Najpopularniejszy jest sklep **Play Google Store** (https://play.google.com/store).

Aby umieścić tam własną aplikację, musisz założyć konto dewelopera w Google Play. Zwykle najpierw trzeba mieć konto w Google. Następnie trzeba się na nie zalogować. Warto zajrzeć na stronę informacyjną dla kandydata na dewelopera:

https://support.google.com/googleplay/android-developer/answer/6112435?hl=pl#zippy=%2Ckrok-za%C5%82%C3%B3%C5%BC-konto-dewelopera-w-google-play.

W prosty i przystępny sposób – krok po kroku – opisano tam, co należy zrobić, aby zostać deweloperem. Między innymi trzeba:

- uzupełnić formularz dewelopera: https://play.google.com/console/signup (w tym podać wszystkie niezbędne dane, jakich będziemy używać do publikacji treści w sklepie Play Google);
- zaakceptować warunki umowy dystrybucji dla programistów;
- wnieść opłatę w wysokości 25 dolarów (USD)[4].

Przygotowanie aplikacji do sprzedaży

Gdy zostaniesz deweloperem Google'a, otrzymasz dostęp do konsoli dewelopera: https://play.google.com/console/about/. W niej będziesz mógł przygotować aplikację do publikacji w sklepie Play24 (https://play.google.com/store).

Inne miejsca umieszczania aplikacji – pliku .apk

Poza wskazanym sklepem Play24 aplikacje (plik .apk) możesz dystrybuować z wykorzystaniem kilku innych miejsc, np.:

- APKPure – https://apkpure.com/pl/;
- Aptoide – https://en.aptoide.com/;
- F-Droid – https://f-droid.org/;
- AppBrain – https://www.appbrain.com/.

Na stronie projektu F-Droid podano: „F-Droid jest instalowalnym katalogiem Wolnego i Otwartego Oprogramowania (ang. *Free and Open Source Software* – FOSS) dla platformy systemowej Android. Aplikacja kliencka ułatwia przeglądanie, instalację i śledzenie uaktualnień na twoim urządzeniu"[5].

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co to jest framework Kivy?
2. Rozwiń skrótowce UI, GUI, WUI i opisz ich znaczenie podczas tworzenia aplikacji.
3. Wymień co najmniej sześć różnych elementów UI aplikacji mobilnej.
4. Co oznacza sformułowanie `** kwargs` w języku programowania Python?
5. Opisz trzy sposoby dodania do pliku .py frameworka Kivy.
6. Wyjaśnij, na czym polega oddzielenie projektowania graficznego aplikacji od kodu tworzącego jej funkcjonalność w języku kv.
7. Na czym polega współpraca frameworka Kivy z relacyjną bazą danych SQL?
8. W jaki sposób możesz poradzić sobie z ewentualnymi problemami, które wystąpią w procesie buildowania?
9. Wskaż miejsca, w których możesz opublikować aplikację. Opisz zasady i warunki, na jakich możesz dystrybuować własne aplikacje.

---

4 Wysokość opłaty na marzec 2021 r. Jedyną formą płatności jest karta płatnicza lub debetowa.

5 Źródło: https://f-droid.org/ (dostęp: 31.03.2021).

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Wykonaj polecenia:
   A. Zainstaluj w swoim komputerze środowisko Python 3.6+.
   B. Stwórz katalog ‘stoper’ i wewnątrz niego zainstaluj środowisko wirtualne języka programowania Python i framework Kivy.
   C. Stwórz plik stoper.py wewnątrz katalogu ‘stoper’. Przygotuj w nim projekt aplikacji Stoper.

2. Wykonaj polecenia:
   Dla posiadaczy komputerów z systemem operacyjnym Windows
   A. Zainstaluj system operacyjny Linux Ubuntu:
      a) wewnątrz VirtualBox (pamiętaj o wcześniejszym sprawdzeniu, czy wirtualizacja na komputerze nie jest zablokowana i czy jest dostępna dla danego procesora) lub
      b) skorzystaj z instalacji ze sklepu Microsoft Store (pamiętaj o wcześniejszym zainstalowaniu technologii WSL2).

   Dla posiadaczy komputerów z systemem operacyjnym Linux i Windows
   A. W systemie operacyjnym Linux Ubuntu zainstaluj aplikację Buildozer.
   B. Za pomocą programu Buildozer przygotuj wersję mobilną na system operacyjny Android programu zapisanego w zadaniu 1 – plik stoper.py.
   C. Jeżeli nie udało się przeprowadzić procesu buildowania, sięgnij po porady zawarte w tym rozdziale podręcznika lub poszukaj informacji w internecie.

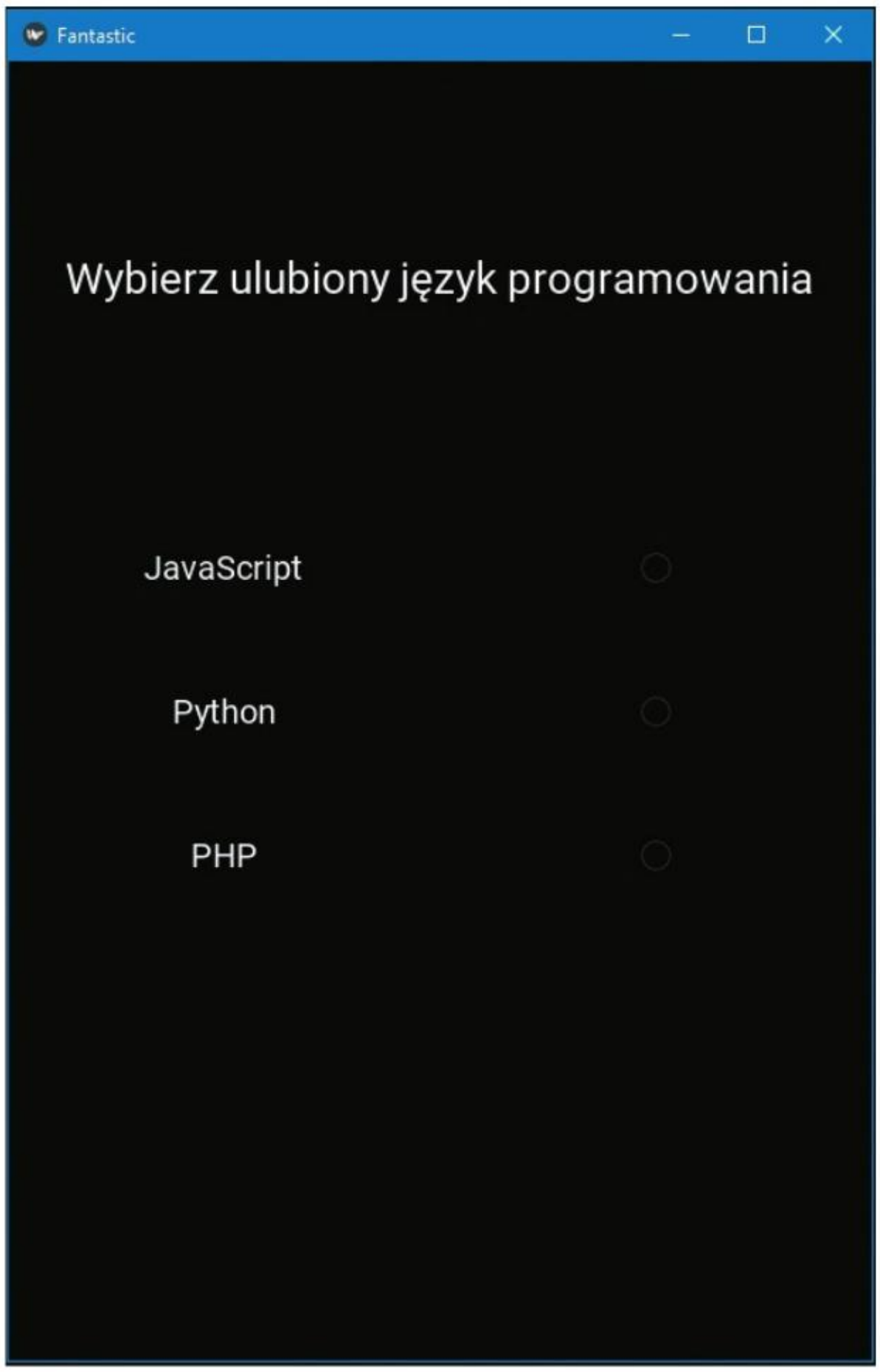
3. Popraw kod Formularza logowania opisanego wcześniej w tym rozdziale, tak aby:
   a) dodać wyświetlanie dla użytkownika brakujących informacji o błędnie wpisanym haśle;
   b) tło aplikacji było jasnozielone.

4. Przygotuj aplikację mobilną z wykorzystaniem elementu UI frameworka Kivy – Accordion. Powinna ona zawierać co najmniej pięć elementów (ekranów). Na każdym z nich umieść zdjęcie np. bohatera filmowego lub książkowego i jego krótki opis. Aplikację przygotuj w dwóch wersjach:
   a) poziomej – trzymaj się kolorystyki zbliżonej do odcieni kolorów: niebieskiego, szarego i białego;
   b) pionowej – zadbaj, aby przeważały kolory: pomarańczowy, czarny i zielony.

5. Jednym z elementów UI dostępnym w Kivy jest checkBox, czyli kontrolka, której zaznaczenie powoduje określone zachowanie wewnątrz aplikacji[6]. Przygotuj ekran aplikacji, który wykorzysta wymieniony element UI. Przygotuj aplikację mobilną, która będzie wyglądała dokładnie tak, jak na rys. 16.72 i 16.73.

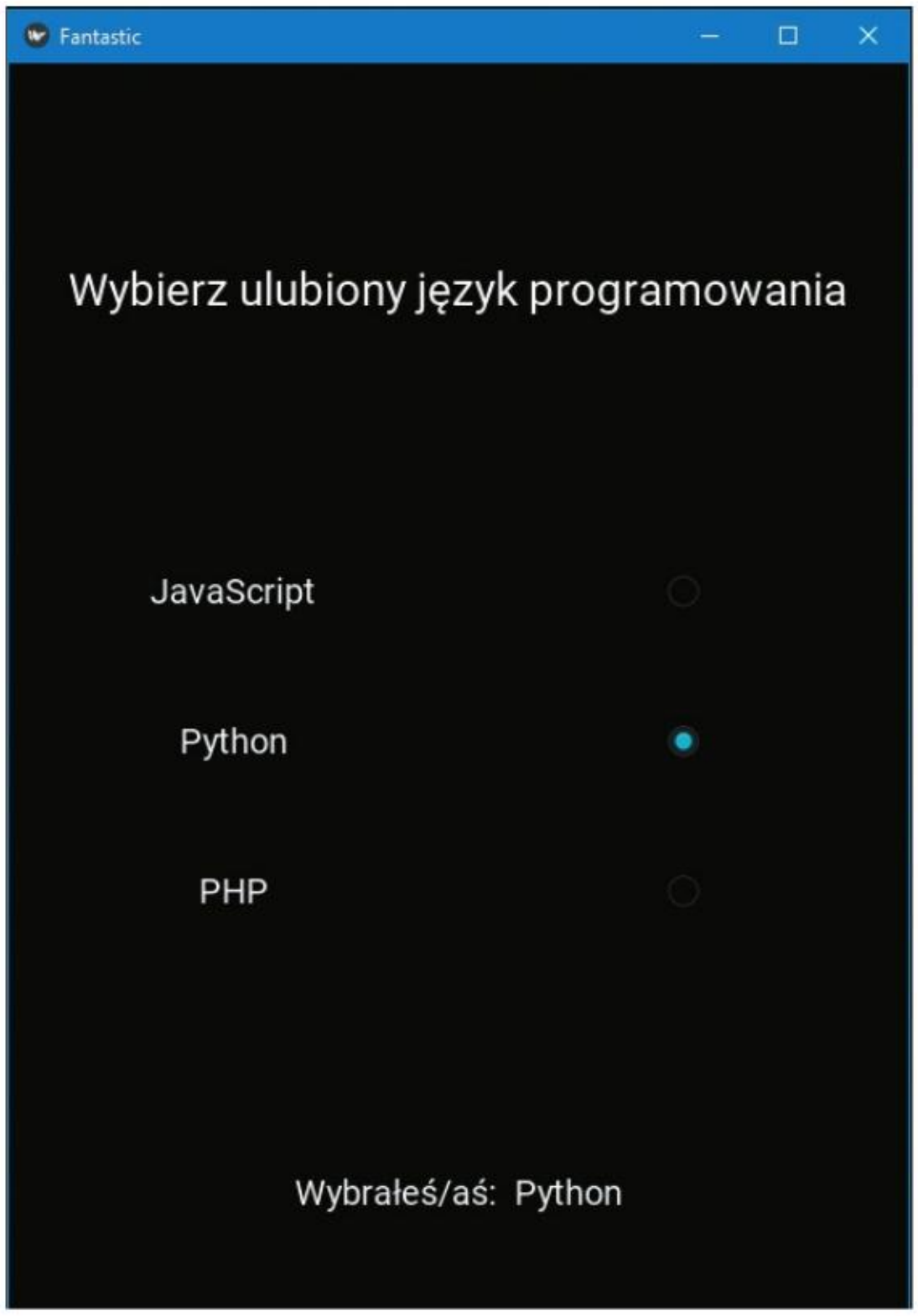
[6] Szczegółowe informacje na wskazany temat możesz znaleźć na stronie: https://kivy.org/doc/stable/api-kivy.uix.checkbox.html.

A. Widok początkowy aplikacji mobilnej – zanim zostanie dokonany wybór za pomocą kliknięcia w jedno z pustych kółek.



Rys. 16.72. Widok początkowy aplikacji mobilnej – przed wyborem

B. Widok aplikacji mobilnej – po wyborze za pomocą kliknięcia w jedno z pustych kółek. (W tym przypadku wybór padł na język programowania Python, ale każdy możliwy wariant zaznaczenia musi zmieniać nazwę języka po sformułowaniu: „Wybrałaś / Wybrałeś").



Rys. 16.73. Widok początkowy aplikacji mobilnej – po wyborze

6. Przygotuj aplikację mobilną, w której:
   A. dwie niewielkie kule będą się wyświetlały jedna pod drugą:
      - czerwona – w lewym górnym rogu,
      - zielona – w lewym dolnym rogu;
   B. po kliknięciu w jedną z nich druga kula będzie wykonywała animację – przemieści się w inne miejsce ekranu:
      - czerwona – w prawy dolny róg,
      - zielona – w prawy górny róg;

      Kula będąca w ruchu musi wrócić do swojego pierwotnego miejsca i znów się przemieścić – zatrzymanie pętli może spowodować dopiero kliknięcie w drugą kulę (opisano to w punkcie D);
   C. kula będąca w ruchu nie może mieć zdarzenia kliknięcia, czyli kliknięcie w nią nie spowoduje żadnej reakcji na stronie;
   D. zatrzymanie kuli będącej w ruchu nastąpi po kliknięciu w drugą kulę, ale nie szybciej niż po jej powrocie do punktu początkowego.

V. Testowanie i dokumentowanie aplikacji

- Walidacja kodu programu
- Tworzenie dokumentacji podczas programowania aplikacji
- Testowanie aplikacji

17 Walidacja kodu programu

ZAGADNIENIA

- Weryfikacja projektu informatycznego
- Narzędzia i środowisko do testowania programów
- Walidacja i obsługa błędów

17.1. Weryfikacja projektu informatycznego – podstawowe pojęcia

Tworząc oprogramowanie użytkowe, bez względu na to, czy jest to kod zwykłej strony internetowej, czy zaawansowana aplikacja webowo mobilna, przychodzi moment oceny jakości wykonanego produktu. To zagadnienie można rozpatrywać w kilku obszarach:

- poprawności działania kodów informatycznych;
- wydajności aplikacji;
- zgodności z oczekiwaniami klienta.

Są to końcowe etapy opracowywania projektu informatycznego, niezbędne w procesie produkcji oprogramowania. Najpierw trzeba jednak wykonać kilka działań, które często określa się jako **weryfikacja wstępna** (lub po prostu weryfikacja).

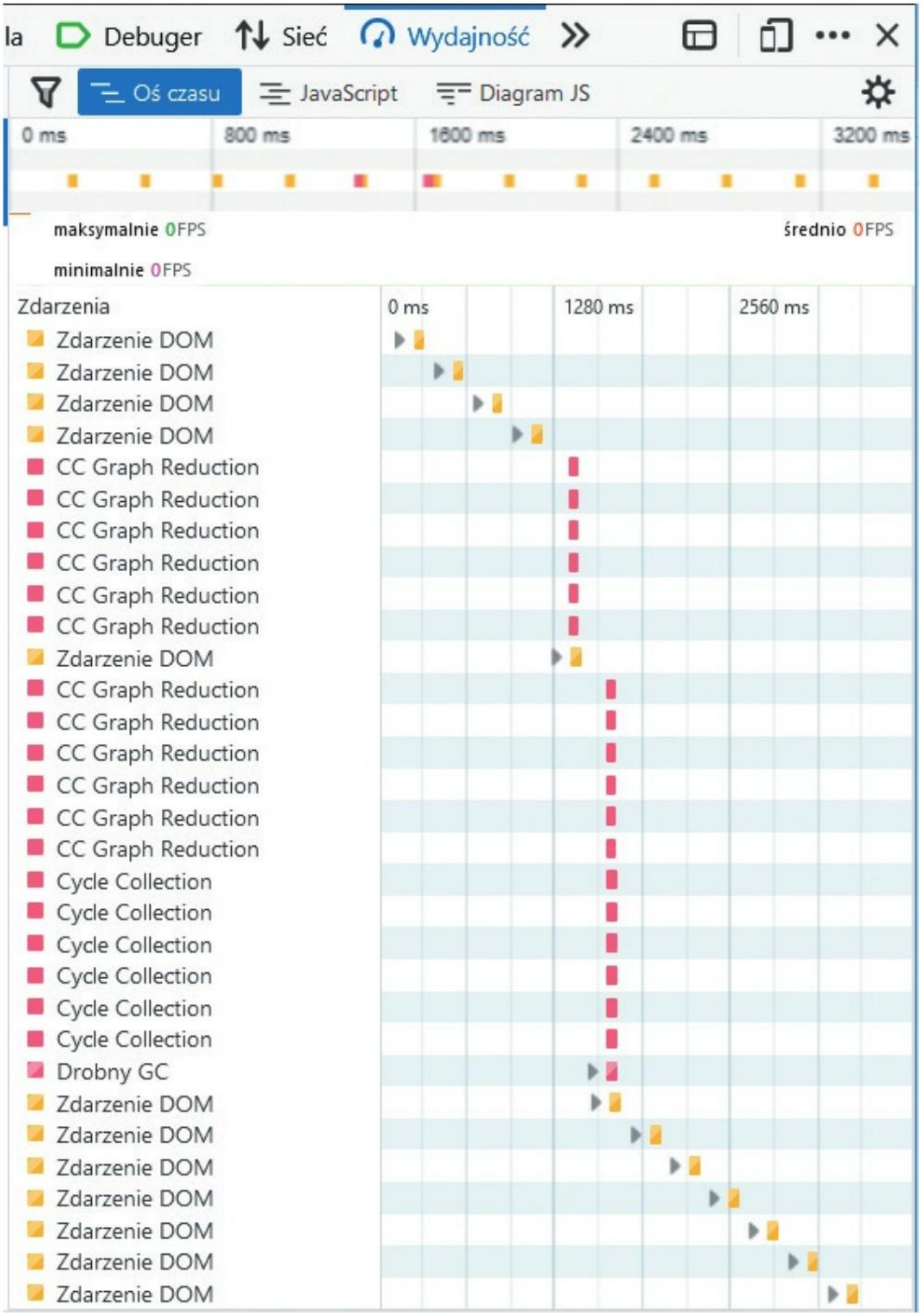
CIEKAWOSTKA

Dzięki korzystaniu z najbardziej popularnych przeglądarek internetowych otrzymujemy w ramach dostarczanego przez nie środowiska technologicznego różnorodne narzędzia dodatkowe, które w codziennej pracy programisty są wręcz niezbędne. Jedno z nich służy do badania wydajności stworzonych stron i aplikacji internetowych. Narzędzie **Wydajność** zapewnia m.in.:

- wgląd w ogólną responsywność witryny;
- obsługę języka JavaScript;
- wydajność układu strony.

Na rys. 17.1 przedstawiono przykład analizy wydajności strony WWW[1].

[1] Zainteresowani zgłębieniem tego tematu powinni odwiedzić stronę https://developer.mozilla.org/en-US/docs/Tools/Performance.



Rys. 17.1. Pasek dewelopera przeglądarki FireFox – zakładka Wydajność

Weryfikacja wstępna

Jest to proces przygotowania i wytworzenia produktu informatycznego, który ma na celu sprawdzenie poprawności całego środowiska, w którym aplikacja będzie tworzona. Może się ona składać z różnych elementów, m.in. zależnie od rodzaju planowanego produktu. Poniższy przykładowy zakres weryfikacji wstępnej oparto na środowisku technicznym (w naszym przypadku jest ono tożsame ze środowiskiem programistycznym).

W skład środowiska technicznego mogą wchodzić:

- infrastruktura techniczna, m.in.:
  - komputery – stacje robocze,
  - sieć IT – internet i intranet,
  - urządzenia niezbędne do testowania – np. podczas produkcji aplikacji mobilnej powinny to być skonfigurowane (m.in. system operacyjny, uprawnienia dostępu, oprogramowanie) urządzenia przenośne (smartfony);
- oprogramowanie narzędziowe;
- języki programowania wraz z właściwym środowiskiem programistycznym;
- systemy operacyjne (zarówno te do pracy na jednostkach roboczych, jak i do testowania);
- personel planowanego projektu (programiści, graficy, menedżerowie i pozostali członkowie zespołu).

Na etapie weryfikacji wstępnej trzeba sprawdzić kompatybilność każdego elementu spośród pięciu wymienionych składowych. Biorąc pod uwagę przygotowywany projekt, należałoby zweryfikować, czy:

- zostały we właściwy sposób dobrane parametry sprzętu do oprogramowania, które będzie na nim używane;
- zatrudnieni programiści (graficy, asystenci itp.) dysponują wiedzą na temat dobranego sprzętu, systemów operacyjnych i planowanej do zastosowania technologii wytwarzania (np. języki programowania);
- urządzenia do testowania mają odpowiednie parametry dotyczące hardware i software (dzięki temu testowanie oprogramowania będzie szybkie i efektywne);
- zostały przygotowane wytyczne dla programistów dotyczące sposobu pisania kodów aplikacji, m.in. użycie tych samych zasad związanych ze sposobem pisania aplikacji – style kodowania, np.:
  - Python – PEP 8, szczegóły: strona domowa projektu: https://pep8.org/;
  - PHP – PSR-12, szczegóły: strona domowa projektu: https://www.php-fig.org/psr/psr-12/;
  - GNU Coding Standards, https://www.gnu.org/prep/standards/standards.pdf;
  - SEI CERT C++ Coding Standard, https://resources.sei.cmu.edu/downloads/secure-coding/assets/sei-cert-cpp-coding-standard-2016-v01.pdf;
  - MISRA C++, https://www.misra.org.uk/MISRACHome/tabid/128/Default.aspx.

Na rys. 17.2 w górnej części pokazano niepoprawne formatowanie, a w dolnej – poprawne, nawiązujące do dobrych praktyk zawartych w PEP 8.

```
# Niewłaściwe sformatowanie kodu
def sprawdzLiczbe(cZgadzaSie, daneInput,
liczbaRuchowInfo, liczbaRuchow, podawaneLiczby,
wpisywaneLiczby):
    print(daneInput)


# Poprawne sformatowanie kodu
def sprawdzLiczbe(
        cZgadzaSie, daneInput, liczbaRuchowInfo,
        liczbaRuchow, podawaneLiczby,
        wpisywaneLiczby):
    print(daneInput)
```

Rys. 17.2. Kod napisany w języku programowania Python

Weryfikacja produkcyjna

Jest to proces sprawdzenia na etapie projektowania i programowania zasad tworzenia projektu (stylu i rodzaju pisania kodów). Na bazie wytycznych opracowanych przed projektem (weryfikacja wstępna) cały zespół projektowy powinien trzymać się określonych zasad tworzenia kodu. W trakcie trwania projektu trzeba systematycznie sprawdzać poprawność kodu względem przyjętych zasad. Takie podejście zagwarantuje spójność kodu tworzonego przez wielu programistów, a także ułatwi utrzymanie aplikacji i jej dalszy rozwój (jej kodów programistycznych).

Jednym ze sposobów zapanowania nad spójnością kodów tworzonych przez wielu programistów jest korzystanie z aplikacji kontroli wersji. Istnieje kilka tego typu rozwiązań, a najpopularniejszym prawdopodobnie jest GIT (strona projektu: https://git-scm.com/). Jeżeli chcemy skorzystać z rozwiązania webowego, to warto polecić bardzo popularny wśród programistów https://github.com/. W pierwszej części podręcznika omówiono oba rozwiązania wraz z powiązaniami między nimi (rozdział 21.3. Systemy kontroli wersji na podstawie Git).



Rys. 17.3. Weryfikacja założeń związanych z realizowanym projektem w stosunku do otrzymanego produktu końcowego – produkt końcowy nie odpowiada początkowym założeniom projektowym

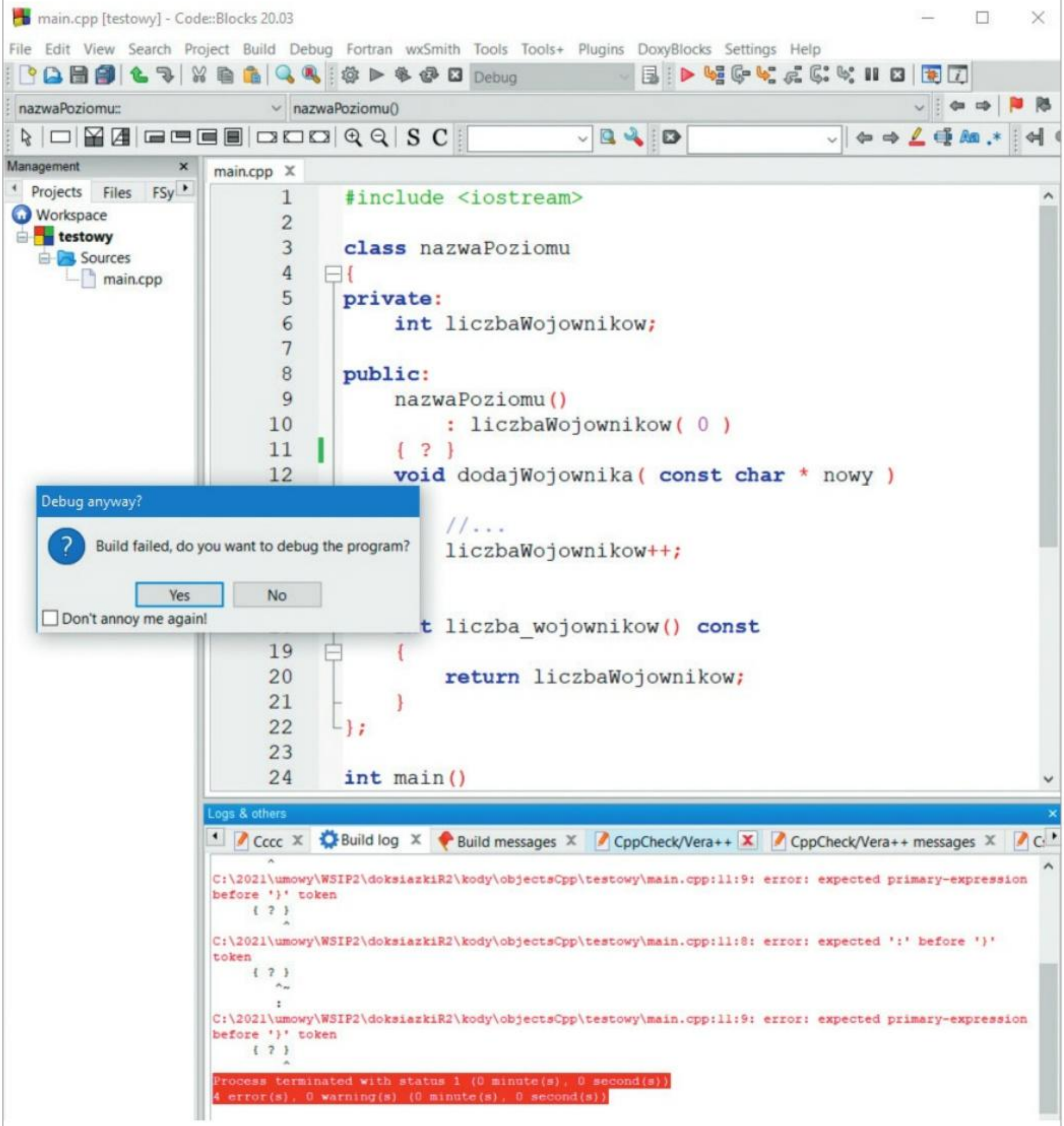
Weryfikacja końcowa

Jest to proces sprawdzenia gotowej wersji produktu informatycznego (np. aplikacji mobilnej) pod względem:

- zgodności z założeniami początkowymi co najmniej w zakresie wyglądu i funkcjonalności.
  Jeżeli mieliśmy do czynienia z wytwarzaniem zwinnym (SCRUM/Agile), to należy dokonać najpierw korekty założeń startowych (uwzględnić zmiany w trakcie kolejnych iteracji projektu). W tym przypadku weryfikacja musi się odnosić do całego procesu wytwarzania zwinnego;
- zgodności z oczekiwaniami klienta.
  Weryfikacja musi być przeprowadzona wspólnie z klientem. Jego uwagi powinny zostać spisane m.in. ze względu na:
  – zapewnienie pełnej satysfakcji klienta z otrzymanego produktu,
  – kolejne projekty (uwagi i reakcje klienta powinny służyć budowaniu strategii rozwoju firmy informatycznej tworzącej projekty);
- zgodności ze stanem prawnym np. w zakresie portali webowych czy aplikacji mobilnych.
  Trzeba uwzględnić aspekty prawne wynikające m.in. z przepisów prawa dotyczących RODO i WCAG 2.1.

17.2. Narzędzia i środowisko do testowania programów

Bardzo ważnym elementem pracy programisty jest znajdowanie i poprawianie błędów w trakcie tworzenia kodu. Na rys. 17.4 przedstawiono etap testowania działania kodu wykonanego w języku programowania C++.



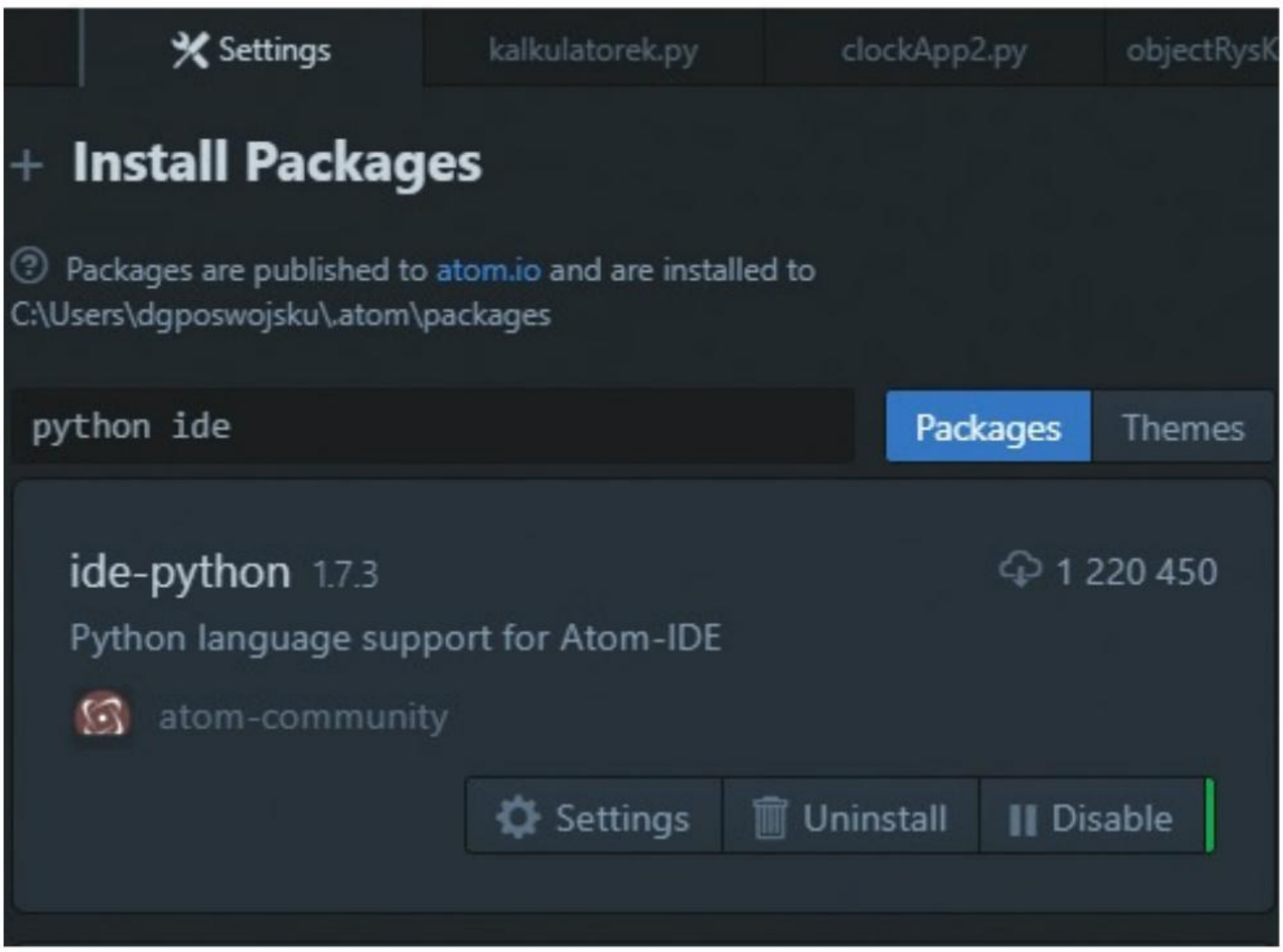
Rys. 17.4. Code::Blocks – testowanie działania kodu wykonanego w języku programowania C++ (informacje o błędach)

Walidacja kodu powinna sprawdzić, czy tworzony kod jest prawidłowy. Ma także za zadanie wskazać poprawność działania kodu. Terminy „walidacja" i „weryfikacja" nieco się przenikają. Dla pewnego uproszczenia w podręczniku termin „walidacja" odnosi się wyłącznie do poprawnego tworzenia kodu, a termin „weryfikacja" – do całej reszty procesu produkcyjnego.

Pojęcie „testowanie" również może być różnie rozumiane. Testuje się np.:

- dopasowanie programu do komputera;
- poprawność działania kodu programistycznego.

Takie podejście sugeruje, że testowanie jest pojęciem bardzo szerokim, zawierającym zarówno weryfikację, jak i walidację. Gdyby przenieść ten termin na poziom stanowiska pracy, to okaże się, że tester w programowaniu zajmuje się tylko jednym obszarem – sprawdzaniem poprawności działania kodu. Trzeba przyznać, że coraz częściej osoba wykonująca ten zawód już nie tylko testuje poprawność działania, lecz także bardzo często wyszukuje konkretne elementy kodu, w których wystąpił błąd. Jeżeli do tego dysponuje odpowiednimi narzędziami informatycznymi i potrafi z nich profesjonalnie korzystać, to może nawet wstępnie poprawiać kod. Poprawki musi jednak zaakceptować programista lub kierownik zespołu.



Rys. 17.5. Program Atom – widok instalowania pakietu ide-python, narzędzia wspomagającego programowanie aplikacji w języku Python

W trakcie pracy programistycznej używa się różnego rodzaju programów wspierających, zwykle określanych jako IDE (lub RAD). Nie tylko podpowiadają one programiście dany kod, lecz także identyfikują błędne elementy już w trakcie pisania kolejnych fragmentów programu. Oznacza to, że zanim jeszcze zostanie podjęta próba debugowania bądź uruchomienia programu, programista widzi na ekranie podkreślone elementy. Dzięki temu wie, że mogą się w nich kryć błędy. Na rys. 17.6 przedstawiono, jak dobry i poprawnie skonfigurowany program IDE potrafi w trakcie pisania wskazywać błędy w kodzie.

```
from kivy.core.window import Window
 kivy.uix.floatlayout import

# Set the app size
Winw.size = błąd

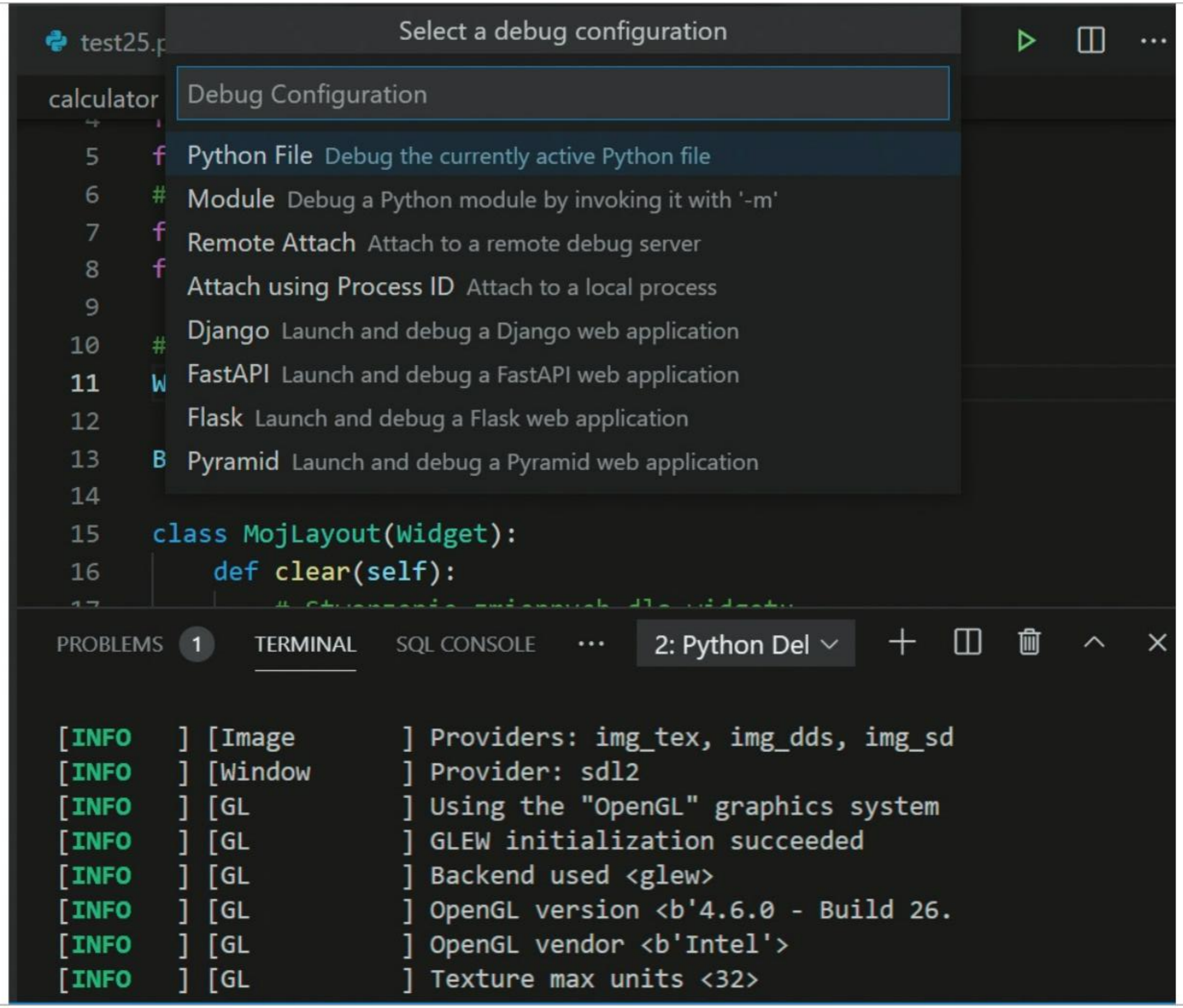
Buir.load_file('

class MojLayout(Widget):
    def clear(self):
        # Stworzenie zmiennych widgetu
        self.ids.calc_input.text = ''
```

Rys. 17.6. Program Visual Studio Code – w trakcie pisania kodu błędy są wskazywane przez program (falowane linie)

Możliwości używanego środowiska programistycznego zwykle można zwiększyć przez zainstalowanie dodatkowych modułów, które wspomogą wykrywanie błędów i ułatwią ulepszanie kodu.

Przykładowo: gdy podczas pracy nad plikiem Pythona uruchomimy proces debugowania, otrzymamy listę modułów do wykorzystania (zainstalowanych wcześniej lub z core IDE).



Rys. 17.7. Visual Studio Code – widok wyboru modułów związanych z debugowaniem napisanego kodu

Mechanizm wskazywania błędów, podobny do przedstawionego na rys. 17.7, istnieje w bardzo wielu popularnych i ogólnie dostępnych programach. Ważne, aby zawsze zwracać uwagę na podpowiedzi o błędach sygnalizowane przez dany program. Przykładowo: na rys. 17.8 przedstawiono panel phpMyAdmin z pakietu XAMPP, który podczas wykonania zapytania SQL do bazy danych również pokazuje błędy.



Rys. 17.8. Panel phpMyAdmin z pakietu XAMPP – wskazanie błędów (pofalowane linie) w kodzie podczas wykonania zapytania SQL do relacyjnej bazy danych

Wybór narzędzi do wychwytywania błędów zarówno w trakcie pisania kodów, uruchamiania, jak i późniejszego ich testowania jest bardzo duży. Od programisty zależy, na jaki zestaw oprogramowania się zdecyduje podczas pracy, natomiast osoba zarządzająca realizacją danego projektu musi podjąć decyzję co do systemowego podejścia w zakresie programowania, w tym etapu, który będzie się odbywał poza stanowiskami pracy programistów.

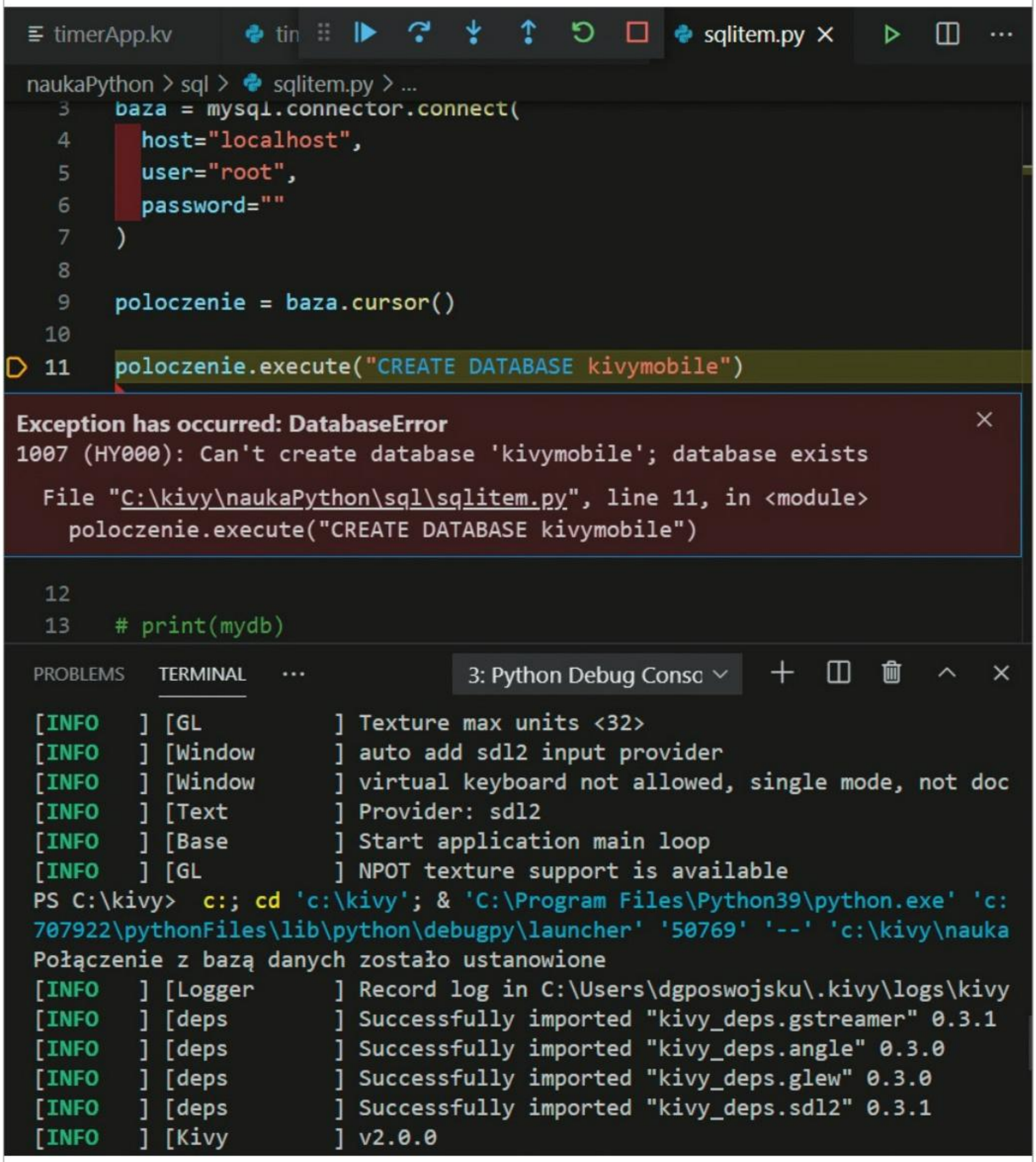
17.3. Walidacja i obsługa błędów

Proces znajdowania błędów w tworzonym oprogramowaniu należy rozpatrywać na kilku poziomach. Najniższy z nich jest bezpośrednio związany z programistą i jego bieżącą pracą nad kodem. O wskazywaniu błędów podczas tworzenia kodu przez program była już mowa.

Mając stosowne oprogramowanie wewnątrz określonego środowiska informatycznego, programista na bieżąco testuje poprawność tworzonego kodu. Robi to przez uruchamianie kolejnych linijek kodu. Dzięki temu jest w stanie sukcesywnie określać skuteczność własnego kodowania. Mamy do czynienia z dwoma sytuacjami:

- testowanie na zasadzie zerojedynkowej – włączam program i albo działa, albo nie działa;
- wspieranie testowania dodatkami programistycznymi, które umożliwiają nie tylko otrzymanie informacji, że np. program nie działa, ale jeszcze dodatkowo odpowiedni

moduł potrafi wskazać (często, lecz nie zawsze), dlaczego program nie chce się uruchomić (lub działa niepoprawnie). Na rys. 17.9 program VSC pokazuje, w którym miejscu w kodzie znajduje się błąd. Jak widać, próbujemy utworzyć bazę danych o nazwie kivymobile, która już istnieje (dodatkowo żółty znacznik po lewej stronie wiersza 11. wskazuje miejsce wystąpienia błędu).



Rys. 17.9. Moduł IDE VSC pokazuje, że program nie zadziała, gdyż już istnieje baza danych o nazwie, którą kod próbuje stworzyć

W trakcie pracy na kodem programistycznym niezmiernie ważne są właściwe, wydajne, dobrze zainstalowane i skonfigurowane narzędzia programistyczne. Wspierają one pracę programisty, pokazują błędy, a bardzo często nawet sugerują rozwiązanie, które można zastosować, by poprawić bądź nawet zoptymalizować stworzony kod.

Co ważne, podczas programowania w Pythonie trzeba niejednokrotnie wybierać między czytelnością kodu a optymalizacją. W trakcie prac nad poprawą wydajności skryptu napisanego w języku programowania Python należy poszukać miejsc problematycznych w skrypcie, pamiętając, że optymalizacja nie zrekompensuje ewentualnych złych decyzji w zakresie zmian w strukturach danych bądź powstania ewentualnych luk w projekcie algorytmu. Jest to bardzo zaawansowany temat, więc optymalizację najlepiej jest potraktować jako naukę czystego kodu. Przechodząc na poziom zaawansowany, można, a nawet trzeba myśleć już kategoriami optymalizacji tworzonego kodu programistycznego (skryptów, struktur danych, powtarzalnych operacji itp.).

Mimo że Python jest językiem interpretowanym, nie oznacza to, iż nie musimy się martwić o optymalizację pod względem szybkości działania.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wyjaśnij, co kryje się pod pojęciem weryfikacji projektu informatycznego.
2. Wymień etapy weryfikacji projektu programistycznego.
3. Opisz, w jaki sposób właściwe oprogramowanie (IDE, RAD) może wspomóc programowanie i jego testowanie.
4. Czy bieżące testowanie kodów ma duże znaczenie? Odpowiedź uzasadnij.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. W utworzonej już aplikacji Stoper usuń część wcięć, np. o jedną spację. Spróbuj uruchomić taki program. Obserwuj systemowe ostrzeżenia, które otrzymasz. Opisz je.

18 Tworzenie dokumentacji podczas programowania aplikacji

ZAGADNIENIA

- Dokumentowanie w kodzie źródłowym programu
- Dokumentacja opisowa kodów programu
- Instrukcja użytkowania aplikacji

18.1. Dokumentowanie w kodzie źródłowym programu

Pamięć ludzka jest bardzo ulotna. Podobnie ma się sprawa z kodem, który dzisiaj piszesz. Jutro wciąż będziesz go pamiętać i bez problemu objaśnisz jego zawiłości, lecz za kilka miesięcy możesz mieć z tym kłopot.

Coraz wyższy poziom umiejętności programistycznych i nabywanie kolejnych doświadczeń zawodowych sprawiają, że podejście do pisania kodów nieustannie się zmienia.

Trzeba pamiętać, że projekt napisany przez jednego programistę może być w przyszłości rozwijany przez zupełnie innego. Najważniejsze, aby zrozumieć sposób kodowania i być w stanie kontynuować czyjąś pracę.

**Aby aplikacja „żyła długo”,
powinna mieć jasno i zrozumiale opisane kody programistyczne.**

Można m.in. umieścić komentarze w kodzie bądź w dokumentacji technicznej.

Komentarz to element kodu, który nie jest interpretowany (pokazywany) przez silniki uruchamiające daną aplikację. Dzięki temu nie zmienia on dla komputera istoty kodu, który został zapisany przez programistę. Jest natomiast niezmiernie cenny dla:

- twórcy aplikacji,
- osób, które w przyszłości będą ją rozwijały.

Stosowanie komentarzy systematycznie opisujących kod programistyczny podnosi zarówno jego jakość, jak i ocenę osoby, która stworzyła kod wraz z opisami.

Na rys. 18.1 przedstawiono kod Pythona i użyty w nim komentarz, na rys. 18.2 znajduje się zaś efekt działania tego kodu (wyświetlenie w konsoli).

```
# Etykieta z tekstem
# Witaj - to jest widoczne!
# wyświetlona zostanie w konsoli
print("Witaj - to jest widoczne!")

# Etykieta z tekstem Witaj Komentarz!
# nie zostanie wyświetlona w konsoli,
# gdyż stoi przed nią znak #
# czyli tak zwany komentarz, który
# ważny jest dla człowieka,
# a równocześnie jest ignorowany
# przez komputer i oprogramowanie
# print("Witaj Komentarz!")

input()
```

Rys. 18.1. Komentowanie w kodzie – język programowania Python

```
(test) PS C:\kivy> & c:/kivy/test/S
cripts/python.exe c:/kivy/uzywaneWP
oradniku/komentowanie.py
Witaj - to jest widoczne!
```

Rys. 18.2. Komentowanie w kodzie – język programowania Python (widok w konsoli)

Wyświetliła się tylko linijka, w której na początku (przed print) nie było znaku #, który oznacza zakomentowanie kodu. Dzięki temu wiemy, że autor nie chce pokazywać tego elementu swojej twórczości. Komentarz ma służyć tylko jemu i innym deweloperom, którzy w przyszłości będą rozwijali daną aplikację.

Spójrz na kilka przykładowych sposobów komentowania w różnych językach programowania:

PHP, JavaScript

```
"""
To jest komentarz wieloliniowy
"""
# To jest komentarz jednoliniowy
```

C, C++, PHP, JavaScript

```
/*
To jest komentarz wieloliniowy
*/
// To jest komentarz jednoliniowy
```

SQL

```
/*
To jest komentarz wieloliniowy - To jest komentarz wieloliniowy -
*/
-- To jest komentarz jednoliniowy
```

HTML

```
<!-
To jest komentarz wieloliniowy
-->
```

CSS

```
/*
To jest komentarz wieloliniowy - To jest komentarz wieloliniowy -
*/
```

```
bind : function() {
    document.querySelector("#button").
    addEventListener("click", () => {
        //jeżeli nie użyjemy funkcji strzałkowej,
        wtedy this wskaże na #button,a nie na ob
        //inną metodą, aby uzyskać opsiywany
        wcześniej efekt jest użycie .bind() lub
        dodatkowej zmiennej const that = this;
        const body = document.querySelector("body");

        this.favoriteColors.forEach(color => {
            const div = document.createElement
            ("div");
            div.style.width = "100px";

            body.appendChild(div);
        });
    });
}
```

Rys. 18.3. Komentowanie opisowe elementów kodu i powiązań – język programowania JavaScript

```
// losowanie liczby z zakresu start do end-1
- reszta z dzielenia
// int wylosowana_liczba =( rand() %
ile_liczb_w_przedziale ) + startowa_liczba;
// https://cpp0x.pl/dokumentacja/standard-C/
srand/587
int wylosowana_liczba = (rand() % end) +
start;
// srand( time( NULL ) );
// Number Of Guess - liczba zgadnięć - do
policzenia
int liczba_zgadniec = 0;
// Wyświetlenie tekstu
std::cout << "Witaj w grze 'Zgadnij Liczbe',
podaj liczbe w zkresie: " << start << " do "
<< end - 1 << std::endl;
```

Rys. 18.4. Komentowanie opisowe elementów kodu i powiązań – język programowania C++

18.2. Dokumentacja opisowa kodów programu

Tworzenie dokumentacji projektowej to prawdziwa sztuka. Dotyczy to zwłaszcza dokumentu przygotowywanego w trakcie realizacji projektu i jego finalnej wersji już po oddaniu aplikacji do użytkowania wewnętrznego, a przed dostarczeniem jej do klienta. Zajmiemy się jednym z aspektów dokumentacji poprojektowej – opisem kodów programistycznych.

Opis w oddzielnym dokumencie powinien być przygotowany w taki sposób, aby zrozumiała go osoba, która nie jest specjalistą od kodów, ale ma podstawową wiedzę techniczną. Tego typu dokument może służyć menedżerom, którzy planują dalszy rozwój aplikacji. Dzięki zrozumiałemu opisowi mogą analizować przydatność (funkcjonalność) poszczególnych kodów bez konieczności angażowania programistów.

W tego typu dokumencie nie tylko opisuje się kody, lecz także mogą być wykazane:

- logiczne powiązania między danym kodem a pozostałą częścią aplikacji;
- elementy podbudowy teoretycznej, które mają pomóc osobom innym niż programiści w zrozumieniu dokumentacji.

W małych i średnich projektach bardzo rzadko sporządza się dokumentację kodów projektu. Tego typu opisy są charakterystyczne przede wszystkim dla bardzo dużych projektów, w których ramach realizuje się zawansowane aplikacje komputerowe.

18.3. Instrukcja użytkowania aplikacji

Gdy stworzysz aplikację i będziesz chciał ją wprowadzić na rynek, będziesz musiał zmierzyć się z opisaniem opracowanego produktu informatycznego, aby każdy:

- chciał się z nim zapoznać;
- zrozumiał zasady funkcjonowania aplikacji.

Aby to było możliwe, trzeba w odpowiedni sposób przygotować pomoc do programu. Najlepiej zrobić to w dwóch wersjach:

- krótkiej – ma ona na celu przedstawienie, głównie w formie graficznej, najważniejszych korzyści, które można uzyskać dzięki aplikacji. Aby przyciągnąć maksymalnie duże zainteresowanie, trzeba pomyśleć nad takim przygotowaniem dokumentu, aby był zajmujący i intrygujący;
- pełnej – w takim dokumencie należy opisać krok po kroku pełną funkcjonalność produktu. Tekst trzeba wesprzeć grafikami.

Żaden z tych dokumentów nie powinien być przygotowany wyłącznie przez programistę. Najlepiej, by wspierały go osoba sprawnie i profesjonalnie posługująca się językiem polskim i grafik, który wzbogaci całość zdjęciami i schematami.

W poprzednich latach istniało kilka formatów plików pomocy (.chm, .htm, .pdf, .md). Obecnie najbezpieczniej jest przygotować taką pomoc w formule interaktywnej, online – w technologii HTML + CSS + JavaScript. Takie podejście zagwarantuje wysoki poziom jakości i funkcjonalności pomocy, a także ułatwi dostęp każdej zainteresowanej osobie. Dobrym pomysłem może być też przygotowanie pliku np. w formacie .odt (Libre Office) bądź .pdf.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Opisz, jak powinno wyglądać dokumentowanie w kodzie źródłowym (programistycznym) programu informatycznego.
2. Scharakteryzuj dokumentację opisową kodów programu.
3. Na jakie elementy należy zwrócić uwagę podczas opracowywania instrukcji użytkowania aplikacji?
4. Czym się różni wersja krótka pomocy do programu od wersji pełnej?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Na końcu rozdziału o aplikacjach mobilnych wykonałeś aplikację z dwoma animowanymi kulami. Przygotuj do niej komentarz nad każdą linijką kodu. Musisz opisać działanie wszystkich wierszy.

19 Testowanie aplikacji

ZAGADNIENIA

- Testowanie w informatyce – podstawy wiedzy
- Testy funkcjonalne i niefunkcjonalne
- Testowanie aplikacji (w tym interfejsu)
- Proces testowania i obsługi błędów w języku programowania Python

19.1. Testowanie w informatyce – podstawy wiedzy

Testowanie oprogramowania to proces mający na celu zapewnienie jego wysokiej jakości. Obejmuje ono:

- weryfikację oprogramowania, która umożliwia kontrolę wytwarzania oprogramowania (zgodności z wcześniej przygotowaną specyfikacją);
- walidację oprogramowania – w tym procesie sprawdza się zgodności oprogramowania z oczekiwaniami (założonymi i końcowymi) użytkownika.

Zależnie od zastosowanej metody testowanie oprogramowania może być wdrożone w dowolnym momencie wytwarzania oprogramowania, np:

- podejście kaskadowe – testowanie zaczyna się wraz z definicją wymagań i jest kontynuowane po zaimplementowaniu zdefiniowanych wymagań;
- metody zwinne (w ostatnich latach głównie to one znajdują zastosowanie w produkcji nowoczesnych aplikacji) – w tym przypadku testowanie jest rozkładane równomiernie na poszczególne iteracje; nacisk kładzie się na testy jednostkowe i automatyzację weryfikacji (testów) wykonywanych przez członków zespołu projektowego.

Istnieje wiele organizacji i instytucji wspierających, nadzorujących i certyfikujących testowanie oprogramowania. Wyznaczają one standardy, które są stosowane przez firmy programistyczne.

Jedną z największych i najbardziej poważanych w tym obszarze firm jest International Software Testing Qualifications Board (ISTQB)[2]. Dysponuje ona m.in. systemem certyfikacji testerów.

[2] Źródło i strona domowa: https://www.istqb.org/.

Efektem systemów certyfikowania testerów oprogramowania jest podwyższenie standardów dla tego zawodu. Firmy produkujące oprogramowanie zrozumiały, że korzystanie z wysoko wykwalifikowanych testerów jest opłacalne i efektywne.

Na rynku istnieje dużo rozwiązań informatycznych, które wspierają proces testowania i wyszukiwania błędów. Warto zapoznać się z dwoma bezpłatnymi do użytku komercyjnego i personalnego:

- Bugzilla – oficjalna strona projektu: https://www.bugzilla.org/;
- Apache JMeter – oficjalna strona projektu: http://jmeter.apache.org.

Bugzilla to oprogramowanie stworzone przez Fundację Mozilla. Za pomocą tego narzędzia można szybko stworzyć serwis internetowy (aplikację internetową) do raportowania błędów w niemal dowolnym oprogramowaniu.

Jest ona wolnym i otwartym oprogramowaniem, co oznacza, że może być używana w projektach zarówno otwartych, jak i komercyjnych.

Działającą aplikację możesz zobaczyć i testować pod adresem https://bugzilla.mozilla.org/home.

W razie potrzeby dysponowania własną instalacją aplikacji trzeba mieć odpowiednie zasoby serwerowe – szczegóły znajdziesz pod adresem https://www.bugzilla.org/requirements/. Są to m.in.:

- dowolna platforma systemowa, na której można zainstalować Perla, w tym Windows, Linux i Mac OS X (Linux nie jest jednak zalecany);
- zasoby sprzętowe – zależą np. od rozmiaru bazy użytkowników i liczby błędów w bazie danych;
- oprogramowanie użytkowe, w tym:
  - Perl (język programowania istniejący ponad 30 lat – szczegóły projektu znajdziesz na jego oficjalnej stronie: https://www.perl.org/);
  - relacyjną bazę danych (np. MySQL, PostgreSQL, Oracle i SQLite);
  - serwer WWW – zaleca się Apache (szczegóły: https://httpd.apache.org/), może także być IIS (Internet Information Services – szczegóły: https://www.iis.net/).

19.2. Testy funkcjonalne i niefunkcjonalne

Testowanie oprogramowania można określić jako sposób oceny funkcjonalności aplikacji, wykonywany w celu wykrycia błędów w stworzonym programie.

Istnieje kilka kryteriów, pod względem których dzieli się testy. Najważniejsze to ich podział na dwa podstawowe rodzaje:

- testy funkcjonalne;
- testy niefunkcjonalne.



Rys. 19.1. Testowanie oprogramowania

Testy funkcjonalne mają na celu zweryfikowane poprawności działania oprogramowania. Wyjaśniają, co robi produkt informatyczny.

Testy niefunkcjonalne (często nazywane **testami pozafunkcjonalnymi**) służą do sprawdzania wydajności oprogramowania. Opisują faktyczne działanie programu.

Testy obciążeniowe, często nazywane także testami wydajnościowymi, należą do grupy testów niefunkcjonalnych. Zwykle są one przeprowadzane zgodnie z ustalonymi scenariuszami potencjalnego użycia produktu informatycznego, np. serwisu internetowego bądź aplikacji mobilnej.

Możliwy schemat testu obciążeniowego:

- **przygotowanie planu testu** – na tym etapie należy ustalić m.in. scenariusz użycia produktu informatycznego (np. aplikacji webowej), liczbę użytkowników testu na określoną jednostkę czasu, w jakiej wykonują oni scenariusz testowania;
- **uruchomienie testu** – monitorowanie zachowania aplikacji w trakcie testowania; dla aplikacji mobilnej można obserwować np. parametry, takie jak:
  - czas ładowania strony projektu,
  - poprawność działania aplikacji,

- stabilność działania serwera,
- zużycie zasobów serwera przez aplikację,
- czas odpowiedzi serwera, na którym „jest postawiona" aplikacja.

Ten rodzaj testowania wykonuje się za pomocą wyspecjalizowanego oprogramowania. Aplikacje webowe można testować np. dzięki darmowej aplikacji JMeter.

„Funkcje Apache JMeter obejmują:

- Możliwość ładowania i testowania wydajności wielu różnych aplikacji / typów serwerów / protokołów:
  - Web – HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, ...),
  - usługi internetowe SOAP / REST,
  - FTP,
  - baza danych za pośrednictwem JDBC,
  - LDAP,
  - oprogramowanie pośredniczące, zorientowane na wiadomości (MOM) za pośrednictwem JMS,
  - poczta – SMTP(S), POP3(S) i IMAP(S),
  - polecenia natywne lub skrypty powłoki,
  - TCP,
  - obiekty Java.
- W pełni funkcjonalny test IDE, który umożliwia szybkie nagrywanie planu testów (z przeglądarek lub aplikacji natywnych), tworzenie i debugowanie.
- Tryb CLI [tryb wiersza polecenia (wcześniej nazywany Non GUI), aby załadować test z dowolnego systemu operacyjnego zgodnego z Java (Linux, Windows, Mac OSX, ...)].
- Kompletny i gotowy do zaprezentowania dynamiczny raportu HTML.
- Łatwą korelację dzięki możliwości wyodrębniania danych z najpopularniejszych formatów odpowiedzi, HTML, JSON, XML lub dowolnego formatu tekstowego.
- Pełną przenośność i 100% czystość kodu Java.
- Pełną strukturę wielowątkowości umożliwiającą równoczesne próbkowanie przez wiele wątków i jednoczesne próbkowanie różnych funkcji przez oddzielne grupy wątków.
- Buforowanie i analizę offline / odtwarzanie wyników testów.
- Wysoce rozszerzalny rdzeń [...]"[3].

Warto pamiętać, że ta aplikacja może służyć także do testowania aplikacji mobilnych opartych na zapleczu bazodanowym.

[3] Źródło: http://jmeter.apache.org (dostęp: 23.03.2021).

Testy bezpieczeństwa systemów informatycznych (w tym aplikacji webowych i mobilnych)

CIEKAWOSTKA

Niektóre dystrybucje Linuksa wyspecjalizowały się w dostarczaniu, wraz z podstawowym systemem operacyjnym, narzędzi służących do testów penetracyjnych sieci. Jednym z najbardziej znanych dla Linuksa jest Kali, oferujący mnóstwo funkcjonalności, które określa się też jako narzędzia hakerskie (bez przesądzania o tym, czy jest to etyczne, czy też przestępcze). Strona projektu: https://www.kali.org/. Na rys. 19.2 znajdziesz widok przykładowych zasobów w podstawowej konfiguracji systemu operacyjnego Kali Linux.



Rys. 19.2. Kali Linux – widok przykładowych narzędzi służących do penetracji sieci

Testy penetracyjne black box (ang. *pentesty black box*) to rodzaj testów funkcjonalnych przeprowadzanych z perspektywy potencjalnego włamywacza – hakera. To oznacza, że tester dysponuje tylko informacjami, które są powszechnie dostępne (publicznie). W przypadku audytu bezpieczeństwa systemu teleinformatycznego pentester nie ma dostępu do architektury danej sieci ani innych dodatkowych informacji wewnętrznych danej firmy. Wyobraź sobie etycznego hakera, który nie ma dodatkowych dostępów poza tymi, które może uzyskać samodzielnie, np. przez rejestrację konta w systemie.

Testy penetracyjne white box (ang. *pentesty white box*) polegają na wykorzystaniu pełnej wiedzy o testowanych zasobach, w tym dokumentacji danego projektu informatycznego. Przeprowadzając audyt bezpieczeństwa stron internetowych i prowadząc tego typu testowanie, etyczny haker otrzymałby dostęp do pełnej dokumentacji i kodu źródłowego aplikacji webowej. Wykonując *pentesty black box* aplikacji mobilnej, tester uzyskałby dostęp do pełnej dokumentacji testowanej aplikacji.

19.3. Testowanie aplikacji (w tym interfejsu)

User Experience (**UX**) to testy użyteczności, które umożliwiają poznanie potrzeb i oczekiwań użytkowników oprogramowania (klientów) celem ich zaspokojenia. Prace z zakresu UX przyspieszają tworzenie funkcjonalnego i użytecznego produktu dla klienta finalnego. Wykonuje się je w różnych fazach produkcji oprogramowania: od pierwszego wyglądu aplikacji przez elementy przejściowe, czyli testowanie kolejnych wersji produktu, aż po testowanie wersji ostatecznej.

Testy użyteczności są ściśle związane z podejściem SCRUM do tworzenia aplikacji. Wraz z każdą kolejną iteracją, przygotowywaną przez zespół pracujący w metodologii programowania zwinnego, do pracy mogą zostać włączeni także testerzy bazujący na User Experience.

Opisany rodzaj testowania, skupiający się przede wszystkim na doznaniach użytkownika systemu, ma w świecie zorientowanym na klienta bardzo duże znaczenie. Skuteczne pilotowanie – obok tworzenia oprogramowania – prac UX może być istotnym elementem prowadzącym do sukcesu biznesowego. Należy dbać o właściwe podejście zespołu projektowego. Podstawowe elementy to:

- najważniejsza jest satysfakcja klienta, wynikająca z używania tworzonego produktu;
- produkt cyfrowy musi być stworzony według zasad:
  - łatwość w obsłudze,
  - wykonywanie założonych działań (funkcjonalność),
  - zgodność z oczekiwaniami konsumenta;
- użyteczność produktu wynika bezpośrednio z potrzeb i oczekiwań klientów, czyli na poszczególnych etapach życia produktu obecność potencjalnych użytkowników końcowych staje się obowiązkowa.

Osiągnięcie sukcesu dzięki prorynkowemu i proklienckiemu podejściu umożliwiają właśnie testy użyteczności.

19.4. Proces testowania i obsługi błędów w języku programowania Python

Systemowe zgłaszanie błędów

Każdy język programowania ma wbudowany mechanizm zgłaszania błędów. Sprawdzimy to na przykładzie Pythona.

Do konsoli systemowej wpisz polecenie uruchamiające środowisko Pythona:

```
python
```

Następnie wpisz polecenie wypisania w konsoli Aga:

- z błędem **`>> pint ('Aga')`**
  - efekt: Python ostrzegł użytkownika, że jest błąd – nie zna polecenia **`'pint'`**;
- poprawnie **`>> print ('Aga')`**
  - efekt: Python wypisał na ekranie oczekiwany napis 'Aga', gdyż zrozumiał polecenie **`'print'`**.

```
>>>
>>> pint ('Witaj Aga')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pint' is not defined
>>>
>>> print ('Witaj, Aga')
Witaj Aga
>>>
>>> print('Witaj, Aga')
Witaj Aga
>>>
```

Rys. 19.3. Konsola CMD, środowisko języka programowania Python – widok polecenia wpisanego błędnie i poprawnie

CIEKAWOSTKA

Na rys. 19.3 możesz zobaczyć, że dla Pythona w konsoli poprawne są polecenia zarówno ze spacją po **`'print'`**, jak i bez niej:

- **`print ('Witaj Aga')`**
- **`print('Witaj Aga')`**

Efekt działania skryptu w obu przypadkach jest taki sam.

W przykładzie na rys. 19.3 widać informacje istotne dla programisty. Python wskazuje:

- NameError – rodzaj błędu;
- miejsce – tutaj wiersz pierwszy, w którym błąd wystąpił.

Identyfikator **`NameError`** jest elementem mechanizmu obsługi „sytuacji wyjątkowych" (lub po prostu wyjątków). Jedną z nich jest wystąpienie błędu w programie.

`NameError` jest identyfikatorem klasy pochodnej od Exception.

Klasa BaseException

W Pythonie jest dostępna bardzo dobra obsługa błędów oparta na klasie Exception. Szczegóły dotyczące tego tematu znajdziesz m.in. na stronie dokumentacji tego języka programowania. Poniżej przedstawiono hierarchię obsługi błędów wbudowaną wewnątrz „silnika Pythona"[4].

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
     +-- StopIteration
     +-- StopAsyncIteration
     +-- ArithmeticError
     |  +-- FloatingPointError
     |  +-- OverflowError
     |  +-- ZeroDivisionError
     +-- AssertionError
     +-- AttributeError
     +-- BufferError
     +-- EOFError
     +-- ImportError
     |  +-- ModuleNotFoundError
     +-- LookupError
     |  +-- IndexError
     |  +-- KeyError
     +-- MemoryError
     +-- NameError
```

[4] Źródło: https://docs.python.org/3/library/exceptions.html?highlight=exception#Exception (dostęp: 31.03.2021).

```
 | +-- UnboundLocalError
 +-- OSError
 | +-- BlockingIOError
 | +-- ChildProcessError
 | +-- ConnectionError
 | | +-- BrokenPipeError
 | | +-- ConnectionAbortedError
 | | +-- ConnectionRefusedError
 | | +-- ConnectionResetError
 | +-- FileExistsError
 | +-- FileNotFoundError
 | +-- InterruptedError
 | +-- IsADirectoryError
 | +-- NotADirectoryError
 | +-- PermissionError
 | +-- ProcessLookupError
 | +-- TimeoutError
 +-- ReferenceError
 +-- RuntimeError
 | +-- NotImplementedError
 | +-- RecursionError
 +-- SyntaxError
 | +-- IndentationError
 |    +-- TabError
 +-- SystemError
 +-- TypeError
 +-- ValueError
 | +-- UnicodeError
 |    +-- UnicodeDecodeError
 |    +-- UnicodeEncodeError
 |    +-- UnicodeTranslateError
+-- Warning
          +-- DeprecationWarning
          +-- PendingDeprecationWarning
```

```
+-- RuntimeWarning
+-- SyntaxWarning
+-- UserWarning
+-- FutureWarning
+-- ImportWarning
+-- UnicodeWarning
+-- BytesWarning
+-- ResourceWarning
```

Zaplanowane zgłaszanie błędów

W języku programowania Python, podobnie jak w innych językach, programista może sam zaplanować obsługę błędów. Wykorzystuje się do tego elementy danego języka, stworzone do takich celów.

Do zaprogramowania własnej obsługi wyjątków może być użyta instrukcja:

try … **except** …

Struktura wygląda następująco:

```
try:
    # Tutaj piszemy kod …
except:
    # Blok opcjonalny
    # Obsługa wyjątków (exception), o ile istnieje taka
    potrzeba
else:
    # Blok zostanie wykonany, gdy nie ma wyjątków
finally:
    # Blok zostanie zawsze wykonany
```

W języku programowania wykonamy prosty kalkulator z jedną funkcją arytmetyczną – dzieleniem. W skrypcie zaprogramujemy obsługę błędów, wykorzystując wiedzę z zakresu:

- instrukcji **try** … **except** …;
- klasy Exception.

```
try:
    wpiszCosLicznik = int(input('Wpisz
    liczbę: '))
    wpiszCosMianownik = int(input('Wpisz
    liczbę: '))
    wynik = wpiszCosLicznik/wpiszCosMianownik

except ArithmeticError:
    print ('Klasa wyjątków arytmetycznych')

except BaseException:
    print ('Klasa podstawowa wyjątków')

else:
    print ("Wpisałeś:  " + str(wynik))

finally:
    print ("Blok finally!")
```

Rys. 19.4. Język programowania Python – kod obsługujący wyjątki

Na rys. 19.4 znajdziesz w blokach:

- **`try`**

  Kod umożliwia pobranie od użytkownika dwóch literałów (dowolnych znaków) i wylicza – dzielenie zmiennej wpiszCosLicznik przez zmienną wpiszCosMianownik. Zauważ, że nie ma żadnego ograniczenia ani nawet wskazówki co do rodzaju znaków, jakie można wpisać. To programiści przewidują, jakie błędy mogą wystąpić, i tworzą kod, który je obsłuży, co oznacza, że mogą wystąpić błędy w postaci próby:

  A. dzielenia przez zero,

  B. dzielenia, w którym licznik i/lub mianownik będą znakiem niebędącym liczbą.

  Ponieważ programista rozpoznał problem potencjalnych błędów (w uproszczeniu opis w punktach A i B), następstwem tego jest zaprogramowanie obsługi potencjalnych błędów – w poniższych dwóch blokach.

- **`except ArithmeticError`**
  Z klasy Exception wybrano ArithmeticError, które obsłuży przechwycenie dzielenia przez zero. Podczas próby dokonania przez użytkownika kalkulatora dzielenia przez zero powstaje błąd, który jest przechwycony i obsłużony w tym bloku. Warto zwrócić uwagę, że blok finally także został wykonany (tak samo jak w kolejnych przypadkach). Na rys. 19.5 przedstawiono wynik działania kodu w konsoli.

```
C:\kodyPython\testowanie>python
 tryExcept.py
Wpisz liczbę: 80
Wpisz liczbę: 0
Klasa wyjątków arytmetycznych
Blok finally!
```

Rys. 19.5. Język programowania Python – widok w konsoli wykonania kodu obsługującego wyjątki (dzielenie przez zero)

Jeżeli jednak błąd byłby inny niż dzielenie przez zero, np. użytkownik wpisze co najmniej jeden znak niebędący liczbą, wówczas ten blok nie zostanie uruchomiony, a program poszuka kolejnego elementu kodu, w naszym przypadku – bloku except BaseException. W takim razie obsługa błędów zostanie przeniesiona do następnego bloku.

- **`except BaseException`**
  W trzecim bloku została użyta klasa nadrzędna wyjątków – możesz wrócić do dołączonego kilka stron wcześniej wykazu klas BaseException i przeanalizować hierarchię. Sięganie od razu po najwyższy poziom niekoniecznie jest właściwe, ale zrobiliśmy tak, aby pokazać cały zakres i zasady działania obsługi błędów w Pythonie.
  Podczas próby wykonania przez użytkownika kalkulatora dzielenia z użyciem innego znaku niż liczba (łącznie z zerem) powstaje błąd, który zostanie przechwycony przez „silnik Pythona" i obsłużony w tym bloku. Wynik działania kodu znajdziesz na rys. 19.6.

```
C:\kodyPython\testowanie>python
 tryExcept.py
Wpisz liczbę: 66
Wpisz liczbę: w
Klasa podstawowa wyjątków
Blok finally!
```

Rys. 19.6. Język programowania Python – widok w konsoli wykonania kodu obsługującego wyjątki (dzielenie z użyciem znaku w mianowniku, który nie jest liczbą)

Także tutaj zadziałała instrukcja finally. Podobnie jak wtedy, kiedy liczbą nie będzie znak wprowadzony w miejsce licznika.

W tym drugim wariancie przejście do bloku BaseException następuje po zatwierdzeniu pierwszego wpisanego ciągu znaków lub znaku.

Wynik działania kodu przedstawiono na rys. 19.7.

```
C:\kodyPython\testowanie>python
 tryExcept.py
Wpisz liczbę: h
Klasa podstawowa wyjątków
Blok finally!

C:\kodyPython\testowanie>python
 tryExcept.py
Wpisz liczbę: string
Klasa podstawowa wyjątków
Blok finally!
```

Rys. 19.7. Język programowania Python – widok w konsoli wykonania kodu obsługującego wyjątki (dzielenie z użyciem znaku w liczniku, który nie jest liczbą)

- **`finally`**
  Instrukcja finally działa dla każdego z opisanych przypadków.

Docstrings – dokumentowanie automatyczne w języku Python

Język programowania Python ma bardzo przydatny mechanizm – **`'docstrings'`** (od ang. *documentation strings*). Jest to narzędzie, które pomaga użytkownikowi lepiej zrozumieć działanie programu.

Ten mechanizm może być szeroko wykorzystywany także w trakcie eksploatacji, gdyż potrafi uzyskać właściwie dowolne – wcześniej przygotowane – informacje, np. opis uruchamianej funkcji.

Na rys. 19.8 przedstawiono przykład wykorzystania **`'docstrings'`**.

```
def wykonajMnozenie(licznik, mianownik):
    '''\
Funkcja dokonuje pomnożenia podanych
dwóch argumentów,o ile oba znaki są
liczbami lub można je zamienić na
liczbę.'''

    try:
        licznik = int(licznik)
        mianownik = int(mianownik)
        print (licznik * mianownik)
    except:
        print ("Jeden z podanych
        argumentów funkcji nie jest
        liczbą")
```

Rys. 19.8. Język programowania Python – widok kodu (górna część) wykorzystującego mechanizm **'docstrings'**

Należy zwrócić uwagę m.in. na użycie potrójnego apostrofu w kodzie. To właśnie w jego wnętrzu są przechowywane informacje, które będą wyświetlane w trakcie działania programu.

```
wykonajMnozenie("as", 6)

print (wykonajMnozenie.__doc__)
```

Rys. 19.9. Język programowania Python – widok kodu (dolna część) wykorzystującego mechanizm **'docstrings'** (w przypadku wpisania znaku, który nie jest liczbą)

```
C:\kodyPython\testowanie>python
 docString.py
Jeden z podanych argumentów fun
kcji nie jest liczbą
Funkcja dokonuje pomnożenia pod
anych dwóch argumentów, o ile ob
ydwa znaki są liczbami lub możn
a je zamienić na liczbę.
```

Rys. 19.10. Język programowania Python – widok wykonania kodu wykorzystującego mechanizm **'docstrings'** (w przypadku wpisania znaku, który nie jest liczbą)

```
wykonajMnozenie(3, 6)

print (wykonajMnozenie.__doc__)
```

Rys. 19.11. Język programowania Python – widok kodu (dolna część) wykorzystującego mechanizm **'docstrings'** (w przypadku wpisania znaku, który jest liczbą)

```
C:\kodyPython\testowanie>python
 docString.py
18
Funkcja dokonuje pomnożenia pod
anych dwóch argumentów, o ile ob
ydwa znaki są liczbami lub możn
a je zamienić na liczbę.

C:\kodyPython\testowanie>
```

Rys. 19.12. Język programowania Python – widok wykonania kodu wykorzystującego mechanizm **'docstrings'** (w przypadku wpisania znaku, który jest liczbą)

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co to jest testowanie produktu informatycznego?
2. Wyjaśnij różnicę między weryfikacją oprogramowania a walidacją oprogramowania.
3. Scharakteryzuj wdrażanie procesu testowania w podejściu kaskadowym i z zastosowaniem metod zwinnych.
4. Co to są testy funkcjonalne i testy niefunkcjonalne? Opisz podstawowe różnice między nimi.
5. Wyjaśnij, na czym polegają i do czego służą testy obciążeniowe.
6. Do jakich celów używa się aplikacji Bugzilla Apache JMeter?
7. Co to są testy penetracyjne?
8. Jaka jest różnica między testami black box a white box.
9. Wyjaśnij, na czym polegają i do czego służą testy użyteczności.
10. Scharakteryzuj pojęcie „systemowe zgłaszanie błędów w języku programowania Python".
11. Co to jest klasa BaseException? Podaj przykłady kilku klas potomnych i je opisz.
12. Co to jest i do czego służy mechanizm `'docstrings'` w języku programowania Python?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Wprowadź korektę w kodzie aplikacji Stoper wykonanej w poprzednim rozdziale w taki sposób, aby po uruchomieniu aplikacji przez użytkownika:
   A. przed 8.00, ale nie wcześniej niż po 4.00, ujrzał on napis: „WITAMY O PORANKU – kalkulator jeszcze nie dział, tylko śpi, wróć po 8.00". Żaden z przycisków nie może być aktywny;
   B. między 8.00 a 20.00 ujrzał on napis „DZIEŃ DOBRY – co chcesz policzyć?". Kalkulator musi działać poprawnie.
   C. między 20.00 a 4.00 ujrzał on napis „DOBRANOC – idź teraz spać – wróć po 8.00". Żaden z przycisków nie może być aktywny.

DODATEK

Kody aplikacji obiektowej Gra edukacyjna Zgadnij Liczbę – framework tkinter

```
import tkinter as tk
from tkinter import ttk
from tkinter.messagebox import showinfo
import random

from PIL import Image, ImageTk

aLiczbaJestWieksza = "\nSzukana liczba jest
większa,\nspróbuj ponownie wpisać jakąś liczbę\n"
bLiczbaJestMniejsza = "\nSzukana liczba jest
mniejsza,\nspróbuj ponownie wpisać jakąś liczbę\n"
cZgadzaSie = "\nBrawo, gratuluję, zgadłeś
liczbę!\nTo faktycznie jest liczba: "

a = 0
b = 99
n = random.randint(a, b)

text_color = "#ffeeee"
bg_color = "#111133"


def global_n_randint(a1, b1):
    global n
    n = random.randint(a1, b1)
```

```
class podaj_liczbe_poczatek:
    def __init__(self):
        self.okno = tk.Toplevel()
        self.okno.wm_title("Podaj  liczbę a")
        self.okno.grab_set()
        self.l = tk.Label(self.okno, text="Podaj
        liczbę początku przedziału losowania")
        self.l.grid(row=0, column=0)

        self.text1 = tk.Entry(self.okno)
        self.text1.grid(row=1,column=0)
        self.b = ttk.Button(self.okno, text="Ok",
        command=self.sprawdzanie_poprawnosci)
        self.b.grid(row=1, column=1)
        # okno.grab_release()
```

```
def sprawdzanie_poprawnosci(self):
    global b
    global a
    t = self.text1.get()
    podany_tekst = t
    try:
        liczba = int(podany_tekst)
        a = liczba
        if a > b:
            popup_showinfo("Błąd","Podana
            liczba jest większa niż górna
            granica.\nNajpierw zwiększ górną
            granicę.")
        else:
            app.l1text.set(
                "\nAby rozpocząć grę,\n wpisz
                liczbę całkowitą z przedziału
                od " + str(a)
                + " do " + str(b) + ",
                \nktóra Twoim zdaniem została
                wylosowana :\n")
            global_n_randint(a,b)
    except:
        popup_showinfo("Błąd", "Podane znaki
        to nie liczba całkowita")
    self.okno.grab_release()
    self.okno.destroy()
```

```
class podaj_liczbe_koniec:
    def __init__(self):
        self.okno = tk.Toplevel()
        self.okno.wm_title("Podaj liczbę b")
        self.okno.grab_set()
        self.l = tk.Label(self.okno, text="Podaj
        liczbę końca przedziału losowania")
        self.l.grid(row=0, column=0)

        self.text1 = tk.Entry(self.okno)
        self.text1.grid(row=1,column=0)
        self.b = ttk.Button(self.okno, text="Ok",
        command=self.sprawdzanie_poprawnosci)
        self.b.grid(row=1, column=1)
```

```
def sprawdzanie_poprawnosci(self):
    global b
    global a
    t = self.text1.get()
    podany_tekst = t
    try:
        liczba = int(podany_tekst)
        b = liczba
        if a > b:
            popup_showinfo("Błąd","Podana
            liczba jest mniejsza niż dolna
            granica.\nNajpierw pomniejsz
            dolną granicę.")
        else:
            app.l1text.set(
                "\nAby rozpocząć grę,\n wpisz
                liczbę całkowitą z przedziału
                od " + str(a)
                + " do " + str(b) + ",
                \nktóra Twoim zdaniem została
                wylosowana :\n")
            global_n_randint(a,b)
    except:
        popup_showinfo("Błąd", "Podane znaki
        to nie liczba całkowita")
    self.okno.grab_release()
    self.okno.destroy()

def popup_showinfo(title="Window", tekst="Hello
World!"):
    showinfo(title, tekst)
```

```
class EntryRozszerzone(tk.Entry):
    def __init__(self, master=None,
    tekst_zastepczy="podaj liczbę", color='grey'):
        super().__init__(master)

        self.tekst_zastepczy = tekst_zastepczy
        self.tekst_zastepczy_color = color
        self.default_fg_color = self['fg']

        self.bind("<Key>", self.focus_in)
        self.bind("<Leave>", self.focus_out)
        self.put_tekst_zastepczy()

    def put_tekst_zastepczy(self):
        self.delete('0', 'end')
        self.insert(0, self.tekst_zastepczy)
        self['fg'] = self.tekst_zastepczy_color

    def focus_in(self, *args):
        if self['fg'] == self.
        tekst_zastepczy_color:
            self.delete('0', 'end')
            self['fg'] = self.default_fg_color

    def focus_out(self, *args):
        if not self.get():
            self.put_tekst_zastepczy()
```

```
class Application(ttk.Frame):
    def __init__(self, master):
        self.z = tk.Frame.__init__(self, master)
        self.pack()
        menubar = tk.Menu(root) # MENU
        filemenu = tk.Menu(menubar,tearoff=0)
        filemenu.add_command(label="Zmień liczbę
        początku przedziału",
        command=podaj_liczbe_poczatek)
        filemenu.add_command(label="Zmień liczbę
        końca przedziału",
        command=podaj_liczbe_koniec)
        filemenu.add_separator()
        filemenu.add_command(label="Exit",
        command=root.quit)
        menubar.add_cascade(label="Program",
        menu=filemenu)
        helpmenu = tk.Menu(menubar, tearoff=0)
        helpmenu.add_command(label="O programie",
        command=lambda: popup_showinfo("O
        programie","Gra 'Zgadnij Liczbę'"))
        helpmenu.add_command(label="O Autorze",
        command=lambda: popup_showinfo("O
        Autorze","Autor: Dariusz Gołębiowski"))
        menubar.add_cascade(label="Pomoc",
        menu=helpmenu)
        root.config(menu=menubar)
```

```
# Górna część okna
self.high_frame = tk.Frame(self.z)
self.high_frame.configure
(background=bg_color)
self.high_frame.pack()

# baner
self.load = Image.open("baner3.png")
self.img = ImageTk.PhotoImage(self.load)
self.l0 = tk.Label(self.high_frame,
image=self.img, bg=bg_color)
self.l0.grid(row=0,column=0,sticky="N")

self.l001 = tk.Label(self.high_frame,
text="Serdecznie witamy!",bg=bg_color,
fg=text_color,font="none 16 bold").grid
(row=1, column=0)

self.l1text = tk.StringVar()
self.l1text.set("\nAby rozpocząć grę,\n
wpisz liczbę całkowitą z przedziału od "
+str(a)+" do "+str(b)+", \nktóra Twoim
zdaniem została wylosowana :\n")
self.l1 = tk.Label(self.high_frame,
textvariable=self.l1text,bg=bg_color,
fg=text_color,font="none 12 bold").grid
(row=2,column=0, sticky="W")
```

```
# środek aplikacji - pobranie informacji
od użytkownika

self.middle_frame = tk.Frame(self.z)
self.middle_frame.configure
(background=bg_color)
self.middle_frame.pack(padx=5)

self.tekst1 = EntryRozszerzone(self.
middle_frame, tekst_zastepczy="podaj
liczbę")
self.tekst1.grid(row=0,column=0)

tk.Label(self.middle_frame,text="",
width=5,bg=bg_color).grid(row=0,column=1)
self.b1 = tk.Button(self.middle_frame,
text="Podaj liczbę",width=9, command=self.
click).grid(row=0,column=2)
```

```
# Wyświetlenie elementów gry
self.low_frame = tk.Frame(self.z)
self.low_frame.configure
(background=bg_color)
self.low_frame.pack()

self.l4text = tk.StringVar()
self.l4text.set("\n\n\n")
self.l2 = tk.Label(self.low_frame,
textvariable=self.l4text,bg=bg_color,
fg=text_color,font="none 16 bold").grid
(row=3,column=0,sticky="W")

# dolna część programu
self.lowest_frame = tk.Frame(self.z)
self.lowest_frame.configure
(background=bg_color)
self.lowest_frame.pack()
l4 = tk.Label(self.lowest_frame,
text="Naciśnij,aby opuścić program ",
bg=bg_color,fg=text_color,font="none 16
bold").grid(row=5,column=0)

tk.Button(self.lowest_frame,
text="Wyjście",width=6, command=self.
close_window).grid(row=5,column=1)
```

```
def click(self):
    global a
    global b
    podany_tekst = self.tekst1.get()
    try:
        liczba = int(podany_tekst)
        if liczba == n:
            liczba = cZgadzaSie + str(liczba)
            + "\n"
            self.l4text.set(liczba)
            MsgBox = tk.messagebox.askquestion
            ('WYGRAŁEŚ','Gratulacje, czy
            chesz zagrać jeszcze raz?')
            if MsgBox == 'yes':
                global_n_randint(a,b)
            else:
                pass
        elif liczba > n:
            liczba = bLiczbaJestMniejsza
        else:
            liczba = aLiczbaJestWieksza
    except:
        liczba = "\nPodane znaki '"
        +podany_tekst+ "' to nie
        liczba\nSpróbuj ponownie.\n"
    self.l4text.set(liczba)
    self.tekst1.put_tekst_zastepczy()
```

```
        def close_window(self):
            root.destroy()
            exit()


root = tk.Tk()
root.title("Zgadnij liczbę")
root.configure(background=bg_color)
app = Application(root)

root.mainloop()
```

Kody aplikacji konsolowej Zgadnij liczbę

```
# Import biblioteki random.py
import random

# Wylosowanie liczby z przedziału 0-99
liczbaStartowa = random.randint(0,99)

# pokazanie wylosowanej liczby
print(liczbaStartowa)

# Zatrzymanie konsoli, żeby zobaczyć wynik losowania
input()

# liczbaStartowa = 55

aLiczbaJestWieksza = "Szukana liczba jest większa, spróbuj ponownie wpisać jakąś liczbę"
bLiczbaJestMniejsza = "Szukana liczba jest mniejsza, spróbuj ponownie wpisać jakąś liczbę"
cZgadzaSie = "Brawo, gratuluję, zgadłeś liczbę! To faktycznie jest liczba: "
liczbaRuchowInfo = "Liczba Twoich ruchów potrzebnych do zgadnięcia właściwej liczby, to: "

tablicaPodawanychLiczb = []

podawaneLiczby = "Podczas zgadywania podałeś liczby: "

daneInput = int(input("Witaj serdecznie. Aby rozpocząć grę, \n wpisz liczbę całkowitą z przedziału od 1 do 99, która Twoim zdaniem została wylosowana : "))

def sprawdzLiczbe(daneInput):

    # Metoda append dołącza do tablicy tablicaPodawanychLiczb cyfrę wpisaną przez użytkownika i przypisaną do zmiennej daneInput
    tablicaPodawanychLiczb.append(str(daneInput))

    # To jest poczatek instrukcji
    if daneInput == liczbaStartowa:
```

```
        # To jest początek tzw. ciała instrukcji
        if
        # len() to funkcja pokazująca liczbę
        elementów umieszczonych w tablicy
        liczbaRuchow = len(tablicaPodawanychLiczb)
        # stworzenie pustego strigna (ciąg znaków)
        wpisywaneLiczby = " "
        # funkcja join(tablica) - zamienia
        tablicę w tzw. stringa
        wpisywaneLiczby = wpisywaneLiczby.join
        (tablicaPodawanychLiczb)

        # pokazywanie w konsoli założonych w
        projekcie wyników
        print(cZgadzaSie, daneInput, ", \n",
        liczbaRuchowInfo, liczbaRuchow, ", \n",
        podawaneLiczby,  wpisywaneLiczby)
        input("Brawo! Zadanie wykonane.\n Kliknij
        klawisz ENTER, aby zamknąć konsolę :).
        ")
        # A to jest koniec pierwszego bloku (tzw.
        ciała) instrukcji if

    # Zaczynamy drugi blok
    elif daneInput < liczbaStartowa:
        # To jest początek drugiego tzw.ciała
        instrukcji if, wykona się ta część kodu,
        gdy nie wykona się instrukcja po if lub
        jak wolisz, gdy wpisana liczba będzie
        mniejsza od liczby wskazanej na początku
        tego prostego algorytmu
        # pokazanie użytkownikowi w konsoli
        informacji o tym, że nie wprowadził
        właściwej liczby
        print(aLiczbaJestWieksza)
        # ponowna prośba o wprowadzenie kolejnej
        (innej) cyfry
        daneInput = int(input('Wpisz inną liczbę
        z przedziału od 1 do 99 : '))
        # uruchomienie funkcji z jej wnętrza,
        czyli tak zwana referencja
        sprawdzLiczbe(daneInput)
```

```
55          # A to jest koniec drugiego bloku (tzw.
            ciała) instrukcji if
56
57      # Zaczynamy trzeci blok
58      else:
59          # To jest początek trzeciego bloku, który
            wykona się jedynie wtedy,gdy nie wykona
            się żaden z pozostałych
60          # pokazanie użytkownikowi w konsoli
            informacji o tym, że nie wprowadził
            właściwej liczby
61          print(bLiczbaJestMniejsza)
62          # ponowna prośba o wprowadzenie kolejnej
            (innej) cyfry
63          daneInput = int(input('Wpisz inną liczbę
            z przedziału od 1 do 99 : '))
64          # uruchomienie funkcji z jej wnętrza,
            czyli tak zwana referencja
65          sprawdzLiczbe(daneInput)
66          # A to jest koniec trzeciego bloku (tzw.
            ciała) instrukcji if
67
68  # Wywołanie funkcji z parametrem
69  sprawdzLiczbe(daneInput)
```

NETOGRAFIA

http://eli.sejm.gov.pl/eli/DU/2019/1861/ogl
http://isap.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU19940240083&type=3
http://jmeter.apache.org
http://mobileangularui.com/
http://www.bloodshed.net/
http://www.prawoautorskie.gov.pl/
http://www.uprp.pl
https://angular.io/
https://apache.org/
https://apkpure.com/pl/
https://beeware.org/
https://clang.llvm.org/
https://code.visualstudio.com/
https://codelite.org/
https://cordova.apache.org/
https://creativecommons.org/
https://dart.dev/
https://developer.android.com
https://developer.android.com/studio
https://developer.apple.com/xcode/
https://developer.mozilla.org/en-US/docs/Tools/Performance
https://developer.mozilla.org/en-US/docs/Web/javascript
https://directory.fsf.org/wiki/License:BSD-2-Clause-FreeBSD
https://docs.microsoft.com/en-us/dotnet/csharp/
https://docs.microsoft.com/en-us/dotnet/desktop/wpf/fundamentals/xaml
https://docs.microsoft.com/pl-pl/dotnet/csharp/
https://docs.microsoft.com/pl-pl/dotnet/desktop/wpf/?view=netdesktop-5.0
https://docs.microsoft.com/pl-pl/dotnet/desktop/wpf/fundamentals/xaml?view=net
desktop-5.0
https://docs.microsoft.com/pl-pl/visualstudio/releases/2019/system-requirements
https://docs.python.org/3/
https://docs.python.org/3/library/exceptions.html?highlight=exception#Exception
https://docs.python.org/3/library/tkinter.html
https://dotnet.microsoft.com/
https://dotnet.microsoft.com/apps/xamarin
https://eli5.readthedocs.io/en/latest/
https://en.aptoide.com/
https://f-droid.org/
https://flutter.dev/

https://framework7.io/
https://gcc.gnu.org/
https://github.com/kivy/buildozer
https://github.com/sumeetchhetri/ffead-cpp
https://glade.gnome.org/
https://httpd.apache.org/
https://ionicframework.com/
https://keras.io
https://kivy.org/
https://kivy.org/doc/stable/guide/licensing.html
https://kivy.org/doc/stable/guide/packaging-android.html#packaging-android
https://kivy.org/doc/stable/guide/packaging-android.html#packaging-your-application-for-the-kivy-launcher
https://kivy.org/doc/stable/guide/packaging-android.html#packaging-your-application--into-apk
https://laravel.com/
https://lightgbm.readthedocs.io/en/latest/index.html
https://mariadb.com/
https://nativescript.org/
https://netbeans.apache.org/
https://nodejs.org/en/
https://numpy.org/
https://opensource.org/licenses/alphabetical
https://opensource.org/licenses/Apache-2.0
https://opensource.org/licenses/BSD-3-Clause
https://opensource.org/licenses/BSL-1.0
https://opensource.org/licenses/LGPL-3.0
https://opensource.org/licenses/MIT
https://opensource.org/licenses/mit-license.php
https://packages.msys2.org/package/mingw-w64-x86_64-glade
https://pandas.pydata.org/
https://pep8.org/
https://pl.espacenet.com/
https://play.google.com/console/signup
https://play.google.com/store
https://pypi.org/project/Theano/
https://python-for-android.readthedocs.io/en/latest/quickstart/
https://pytorch.org/
https://reactnative.dev/
https://sklearn.org/
https://sourceforge.net/projects/mingw/files/

https://sqlite.org/index.html
https://sqlitebrowser.org/
https://stroustrup.com/compilers.html
https://stroustrup.com/index.html
https://support.google.com/googleplay/android-developer/answer/6112435?hl=pl#zippy=%2Ckrok-za%C5%82%C3%B3%C5%BC-konto-dewelopera-w-google-play
https://symfony.com/
https://ubuntu.com/#download
https://uprp.gov.pl/pl/publikacje/publikacje-bezplatne
https://visualstudio.microsoft.com/pl/
https://visualstudio.microsoft.com/pl/downloads/
https://wiki.python.org/moin/TkInter
https://www.apachefriends.org/pl/index.html
https://www.appbrain.com/
https://www.apple.com/
https://www.bugzilla.org/
https://www.codeblocks.org/
https://www.codeblocks.org/downloads/
https://www.cygwin.com/
https://www.debian.org/index.en.html
https://www.eclipse.org/
https://www.embarcadero.com/products/rad-studio
https://www.epo.org/searching-for-patents/technical/espacenet.html
https://www.fsf.org/
https://www.gnu.org/licenses/license-list.html#SoftwareLicenses
https://www.gnu.org/philosophy/free-sw.html
https://www.google.com/
https://www.iis.net
https://www.istqb.org/
https://www.jetbrains.com/idea/
https://www.microsoft.com/pl-pl/software-download/windows10
https://www.perl.org/
https://www.php.net/
https://www.php-fig.org/psr/psr-12/
https://www.phpmyadmin.net/
https://www.pyinstaller.org/
https://www.python.org/
https://www.qt.io/
https://www.qt.io/download
https://www.qt.io/product/framework
https://www.scipy.org/

https://www.sencha.com/products/touch/
https://www.telerik.com/kendo-ui
https://www.tensorflow.org/
https://www.virtualbox.org/
https://www.w3.org/standards/webdesign/htmlcss
https://www.wipo.int/portal/en/index.html

WYKAZ PODSTAWOWYCH POJĘĆ W JĘZYKACH: POLSKIM, ANGIELSKIM I NIEMIECKIM

JĘZYK POLSKI	JĘZYK ANGIELSKI	JĘZYK NIEMIECKI
aktualizacja	update	die Aktualiesierung
algorytm	algorithm	der Algorithmus
analiza	analysis	die Analyse
architektura klient–serwer	client-server model	das Klient-Server-Modell
archiwizacja	data archiving	die Datenversicherung / die Archivierung
atrybut	attribute	das Attribut
baza danych	database	die Datenbank
bezpieczeństwo danych	data safety	die Datensicherheit
biznesplan	business plan	der Geschäftsplan/ Businesplan
blok tekstu	block of text	der Textblock
bramka logiczna	logic gate	die Logic-Gatter
czas pracy	work time	die Arbeitszeit
czcionka	font	die Schriftart
dane osobowe	personal data	die Personaldaten
dokument	document	das Dokument / die Urkunde
filtr	filter	der Filter
formatowanie	formatting	die Formatierung
formularz	form	das Formular
funkcja	function	die Funktion
grafika rastrowa	raster graphics	die Bitmapgrafik / die Rastergrafik / die Pixelgrafik
grafika wektorowa	vector graphics	die Vektorgrafik
grupa robocza	workgroup	die Arbeitsgruppe
hasło	password	das Passwort
hierarchia	hierarchy	die Hierarchie
hipertekstowy język znaczników	hypertext markup language	die Hypertext-Auszeichnungssprache / die Web-Programmiersprache
host	host	der Host
ikonka	icon	die Ikone
implementacja	implementation	die Implementierung

JĘZYK POLSKI	JĘZYK ANGIELSKI	JĘZYK NIEMIECKI
indeksowanie	indexing	die Indexierung
instrukcja obsługi	manual	das Handbuch
interfejs	interface	das Interface
iteracja	iteration	die Iteration / die Wiederholung
język skryptowy	scripting language	die Scriptsprache
język SQL	SQL language	die Strukturierte Abfragesprache
kaskadowy arkusz stylów	cascading style sheet	der Kaskadenstilentwurf
klient	client	der Kunde / der Benutzer
kolorystyka	colour scheme	die Farbzusammenstellung
konkatenacja	concatenation	die Verkettung
konkurencja	competition	die Konkurrenz
konto użytkownika	user account	das Benutzerkonto
kosztorys	estimated budget / cost estimation	die Berechnung / der Kosten-voranschlag
kwalifikacje zawodowe	professional qualifications	die Berufsqualifikationen
licencja	licence	die Lizenz
logowanie	login	die Anmeldung
marketing	marketing	das Marketing
menu nawigacyjne	navigation menu	das Navigationsmenü
metoda	method	die Methode
miejsce pracy	work place	der Arbeitsplatz
motyw	theme	das Motiv
obiekt	object	das Objekt
obliczenie	computation	die Berechnung
obraz cyfrowy	digital image	das digitale Bild
ochrona	protection	der Schutz
ograniczenie	limitation	die Begrenzung
organizacja pracy	organisation of work	die Arbeitsplanung
pętla	loop	die Schleife
planista	planner	der Planer
poczta elektroniczna	e-mail	elektronische Post
powiadomienie	notification	die Mitteilung
praca zdalna	remote work	die Heimarbeit

JĘZYK POLSKI	JĘZYK ANGIELSKI	JĘZYK NIEMIECKI
pracownik	worker / employee / laborer	der Arbeiter
prawo autorskie	copyright	das Urheberrecht
priorytet	priority	das Priorität
proces	process	der Prozess
profil użytkownika	user profile	das Benutzerprofil
programista	coder / developer	der Programmierer
projekt	project	das Projekt
prośba	request	die Anfrage
protokoły	protocols	die Protokolle
przeglądarka	browser	der Browser
przepis prawny	regulation	die Rechtsverordnung
przycisk	button	der Knopf / die Taste
przyjazny dla użytkownika	user-friendly	anwenderfreundlich / benutz-erfreundlich
relacyjna baza danych	relational database	die relationale Datenbank
roboczogodzina	man-hour	die Betriebsstunde
rozdzielczość	resolution	die Auflösung
sekwencja	sequence	die Sequenz
skalowanie	calibration	das Skalieren / die Skalierung
stanowisko pracy	work station	der Arbeitsplatz
strukturalny język zapytań	the structured language of queries	die Strukturierte Abfragesprache
system bazy danych	database system	das Datenbanksystem
system zarządzania bazą danych	database management system	das Datenbankverwaltungssystem
tabela	table	die Tabelle
testowanie	testing	der Softwaretest
typ danych	data type	der Datentyp
układ graficzny	typography	die Typografie
walidacja	validation	die Validierung
wiersz tabeli	table row	die Tabellenzeile
witryna internetowa	website	die Website
właściwości pola	field properties	die Feldeigenschaften
właściwości tabeli	table properties	die Tabelleneigenschaften
wprowadzenie na rynek	launch	der Launch / der Markteintritt

JĘZYK POLSKI	JĘZYK ANGIELSKI	JĘZYK NIEMIECKI
współpraca	cooperation	die Zusammenarbeit
współpracownik	colleague	der Mitarbeiter
wykaz	list	die Liste
wyszukać i usuwać usterki	to debug	die Fehler beseitigen (ein Programm debuggen)
zapis (rekord)	record	das [Daten]satz
zapytanie (kwerenda)	query	die Abfrage
zasada	principle	das Prinzip
zawód	profession	der Beruf
zdarzenie	happening	das Ereignis
zmienna	variable	die Variable
znak interpunkcyjny	punctuation mark	das Satzzeichen
zorientowany obiektowo	object-oriented	objektorientiert
żargon	jargon	der [Fach]jargon

Źródła ilustracji

Okładka: aplikacje (Production Perig/Shutterstock.com)
Tekst główny: s. 32 (płyta) Nor Gal/Shutterstock.com; s. 39 (elementy środowiska programistycznego) Vladgrin/Shutterstock.com; s. 78 (szklanka) gnoparus/Shutterstock.com; s. 80 (logo gry) Volonoff/Shutterstock.com; s. 100 (ryś) Henryk Sadura/Shutterstock.com; s. 229–230 (ryś) Henryk Sadura/Shutterstock.com; s. 236 (kot) Dariusz Gołębiowski; s. 239 (kot) Dariusz Gołębiowski; s. 276 (efekt końcowy działania Buildozera) Dariusz Gołębiowski; s. 279 (widok instalacji pliku) Dariusz Gołębiowski; s. 280 (widok potwierdzenia zainstalowania pliku) Dariusz Gołębiowski; s. 290 (krajobraz) Andrzej Puchta/Shutterstock.com, (logo parku) 21graphic/Shutterstock.com, (laptop) Passatic/Shutterstock.com

Wydawnictwa Szkolne i Pedagogiczne oświadczają, że podjęły starania mające na celu dotarcie do właścicieli i dysponentów praw autorskich wszystkich zamieszczonych utworów. Wydawnictwa Szkolne i Pedagogiczne, przytaczając w celach dydaktycznych utwory lub fragmenty, postępują zgodnie z art. 27^1 ustawy o prawie autorskim. Jednocześnie Wydawnictwa Szkolne i Pedagogiczne oświadczają, że są jedynym podmiotem właściwym do kontaktu autorów tych utworów lub innych podmiotów uprawnionych w wypadkach, w których twórcy przysługuje prawo do wynagrodzenia.