owoce programowania

Helion

>>> PYTHON

DLA ZUPEŁNIE POCZĄTKUJĄCYCH

WYDANIE IV

TONY GADDIS

Tytuł oryginału: Starting Out with Python (4th Edition)

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-4682-6

HELION SA
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/pyzup4.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/pyzup4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 13

Programowanie GUI

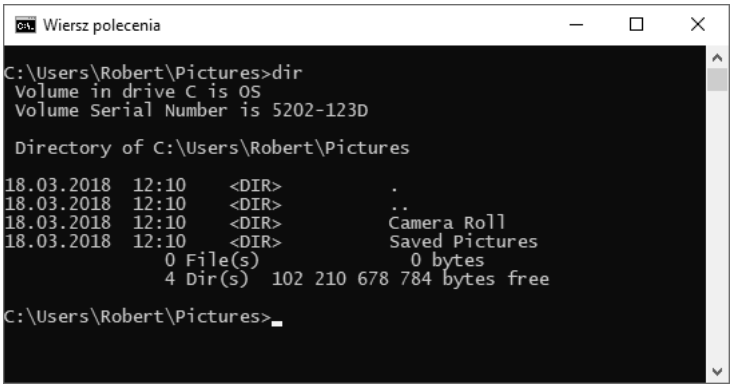
TEMATYKA

13.1. Graficzny interfejs użytkownika

WYJAŚNIENIE. **Graficzny interfejs użytkownika pozwala komunikować się z systemem operacyjnym i programami za pomocą elementów graficznych, takich jak ikony, przyciski czy okna dialogowe.**

Interfejs użytkownika to ten element, z użyciem którego użytkownik komunikuje się z komputerem. Jednymi ze składników interfejsu użytkownika są urządzenia, takie jak klawiatura czy monitor. Innym jest sposób, w jaki komputer przyjmuje od użytkownika polecenia. Przez bardzo wiele lat jedynym sposobem komunikacji użytkownika z komputerem był **interfejs wiersza poleceń**, taki jak pokazany na rysunku 13.1. Interfejs ten zazwyczaj wyświetla znak zachęty — po nim użytkownik może wprowadzić polecenie, które następnie wykona komputer.

Wielu użytkownikom komputerów, zwłaszcza początkującym, interfejs wiersza poleceń wydaje się zbyt skomplikowany. Wynika to z tego, że aby z niego korzystać, należy poznać wiele poleceń, z których każde ma inną składnię — podobnie jak w językach programowania. Jeśli podczas wprowadzania polecenia użytkownik popełni błąd, polecenie nie zadziała.



Rysunek 13.1. Przykład interfejsu wiersza poleceń

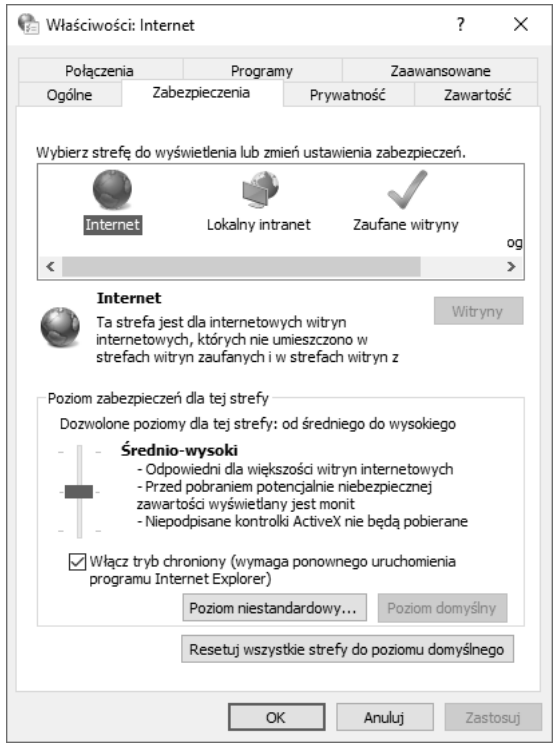
W latach 80. ubiegłego wieku w komercyjnych systemach operacyjnych pojawił się nowy typ interfejsu — **graficzny interfejs użytkownika** (ang. *graphical user interface* — **GUI**). Korzystając z graficznego interfejsu użytkownika, można się komunikować z komputerem za pomocą elementów graficznych wyświetlanych na ekranie. GUI przyczynił się także do spopularyzowania myszy komputerowej jako urządzenia wskazującego. Zamiast wprowadzać polecenia przy użyciu klawiatury, użytkownik może wskazać dany element na ekranie i aktywować go naciśnięciem przycisku myszy.

W przypadku GUI za interakcję z użytkownikiem w dużej mierze odpowiadają **okna dialogowe**, czyli okna, w których wyświetlają się informacje i które umożliwiają użytkownikowi wykonanie określonych operacji. Na rysunku 13.2 przedstawiłem przykładowe okno dialogowe, z pomocą którego użytkownik systemu operacyjnego Windows może zmodyfikować opcje internetowe. Zamiast wprowadzać tajemnicze polecenia, użytkownik korzysta z elementów graficznych, takich jak ikony, przyciski czy suwaki.

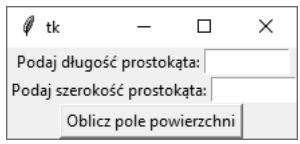
Programy z GUI to programy sterowane zdarzeniami

W środowiskach tekstowych, czyli np. w interfejsie wiersza poleceń, to program decyduje o tym, w jakiej kolejności będą wykonane polecenia. Wyobraź sobie program obliczający pole powierzchni prostokąta. Najpierw program poprosi użytkownika o podanie długości pierwszego boku prostokąta. Użytkownik wprowadza wartość, po czym program prosi go o podanie długości drugiego boku prostokąta. Użytkownik wprowadza wartość, a program oblicza pole powierzchni prostokąta. Użytkownik nie ma wpływu na to, w jakiej kolejności będzie podawał dane.

W przypadku środowiska wyposażonego w GUI to użytkownik wybiera kolejność operacji. Przykładowo na rysunku 13.3 pokazałem wyposażony w GUI program (utworzony w Pythonie) obliczający pole powierzchni prostokąta. Użytkownik może w tym przypadku podawać długości boków w dowolnej kolejności. Jeśli popełni błąd, może w każdej chwili usunąć wprowadzone dane i wpisać poprawne. Gdy użytkownik chce, aby program obliczył pole powierzchni prostokąta, klika przycisk *Oblicz pole powierzchni*. Ponieważ programy wyposażone w GUI muszą reagować na działania



Rysunek 13.2. Przykład okna dialogowego



Rysunek 13.3. Przykład programu wyposażonego w GUI

wykonywane przez użytkownika, nazywa się je programami **sterowanymi zdarzeniami**. Użytkownik, klikając przycisk, powoduje wystąpienie w programie określonego zdarzenia, a program musi to zdarzenie obsłużyć.

Punkt kontrolny

13.1. Do czego służy interfejs użytkownika?

13.2. W jaki sposób działa interfejs wiersza poleceń?

13.3. Co decyduje o kolejności operacji w przypadku programu działającego w środowisku tekstowym, takim jak interfejs wiersza poleceń?

13.4. Co to są programy sterowane zdarzeniami?

13.2. Używanie modułu tkinter

WYJAŚNIENIE. **Do utworzenia prostych programów z użyciem GUI w Pythonie można skorzystać z modułu `tkinter`.**

Język Python nie ma wbudowanych możliwości przeznaczonych do tworzenia graficznego interfejsu użytkownika. Jednak jest dostarczany wraz z modułem o nazwie `tkinter` pozwalającym na tworzenie prostych programów wyposażonych w GUI. Nazwa „tkinter" jest skrótem od „Tk interface". Moduł ten umożliwia programistom Pythona użycie biblioteki graficznej o nazwie Tk. Wiele innych języków programowania również używa biblioteki Tk.

UWAGA. Istnieje wiele innych bibliotek GUI dostępnych dla Pythona. Skoro moduł `tkinter` jest dostarczany wraz z Pythonem, w tym rozdziale do tworzenia programów wyposażonych w graficzny interfejs użytkownika będę korzystał tylko z tego modułu.

Program wyposażony w GUI wyświetla okno wraz z różnymi graficznymi **widżetami**, które użytkownik może wykorzystać do pracy z programem. Moduł `tkinter` dostarcza 15 widżetów, które wymieniłem w tabeli 13.1. Wprawdzie w rozdziale nie omówię ich wszystkich, ale pokażę, jak można utworzyć prosty program wyposażony w GUI, służący do pobierania i wyświetlania danych.

Najprostszy możliwy program wyposażony w GUI wyświetla puste okno. Na listingu 13.1 przedstawiłem kod takiego programu utworzonego za pomocą modułu `tkinter`. Po uruchomieniu programu zobaczysz okno pokazane na rysunku 13.4. Aby zakończyć działanie programu, wystarczy po prostu kliknąć standardowy przycisk zamknięcia programu w Windows (*x*), w prawym górnym rogu okna.

UWAGA. Programy oparte na module `tkinter` nie zawsze działają niezawodnie z poziomu środowiska IDLE. Wynika to z tego, że IDLE samo w sobie korzysta z `tkinter`. Wprawdzie edytor IDLE zawsze można wykorzystać do utworzenia kodu źródłowego programów wyposażonych w GUI, ale najlepsze wyniki daje ich uruchamianie z poziomu wiersza poleceń systemu operacyjnego.

W wierszu 3. został zaimportowany moduł `tkinter`. W definicji funkcji `main()` w wierszu 7. następuje utworzenie egzemplarza klasy `Tk` modułu `tkinter` i przypisanie go zmiennej `main_window`. Obiekt ten jest widżetem głównym, czyli oknem głównym programu. W wierszu 10. znajduje się wywołanie funkcji `mainloop()` modułu `tkinter`. Funkcja jest do chwili zamknięcia okna jak pętla działająca w nieskończoność.

Podczas tworzenia programów wyposażonych w GUI większość programistów preferuje podejście zorientowane obiektowo. Zamiast przygotowywać funkcję tworzącą wyświetlane na ekranie elementy programu, powszechną praktyką jest zdefiniowanie

Tabela 13.1. Widżety oferowane przez moduł tkinter

Widżet	Opis
`Button`	Przycisk, którego kliknięcie może wywołać pewną akcję.
`Canvas`	Prostokątny obszar, który może być używany do wyświetlania grafiki.
`Checkbutton`	Przycisk, który może być w stanie „włączony" lub „wyłączony".
`Entry`	Obszar, w którym użytkownik może za pomocą klawiatury podać jeden wiersz danych wejściowych.
`Frame`	Kontener, który może przechowywać inne widżety.
`Label`	Obszar wyświetlający jeden wiersz tekstu lub obraz.
`Listbox`	Lista, z której użytkownik może wybierać element.
`Menu`	Lista opcji menu wyświetlanych po kliknięciu widżetu `Menubutton`.
`Menubutton`	Menu wyświetlane na ekranie; użytkownik może je klikać.
`Message`	Obszar pozwalający na wyświetlenie wielu wierszy tekstu.
`Radiobutton`	Widżet, który może być wybrany lub nie. Taki widżet zwykle pojawia się w grupie i pozwala użytkownikowi na wybór jednej z kilku opcji.
`Scale`	Widżet pozwalający użytkownikowi na wybór wartości przez przesunięcie suwaka.
`Scrollbar`	Ten widżet może być używany wraz z innymi typami, aby zapewnić możliwość przewijania.
`Text`	Widżet pozwalający użytkownikowi na wprowadzenie wielu wierszy tekstu danych wejściowych.
`Toplevel`	Kontener, podobny do `Frame`, ale wyświetlany we własnym oknie.

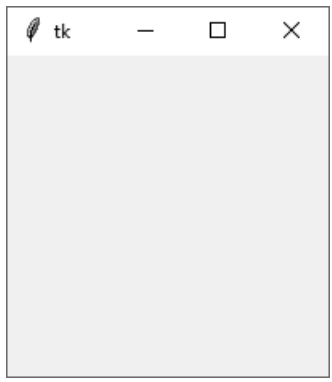
Listing 13.1 (empty_window1.py)

```
1  # Ten program wyświetla puste okno.
2
3  import tkinter
4
5  def main():
6      # Utworzenie widżetu okna głównego.
7      main_window = tkinter.Tk()
8
9      # Wejście do pętli głównej tkinter.
10     tkinter.mainloop()
11
12 # Wywołanie funkcji main().
13 main()
```

klasy wraz z metodą `__init__()` generującą graficzny interfejs użytkownika. Po utworzeniu egzemplarza klasy na ekranie zostaje wyświetlony GUI. Na listingu 13.2 przedstawiłem zorientowaną obiektowo wersję programu wyświetlającego puste okno. Po jego uruchomieniu zostanie wyświetlone puste okno, tak jak pokazałem wcześniej na rysunku 13.4.

Listing 13.2 (empty_window2.py)

```
1  # Ten program wyświetla puste okno.
2
3  import tkinter
4
5  class MyGUI:
6    def __init__(self):
7      # Utworzenie widżetu okna głównego.
8      self.main_window = tkinter.Tk()
9
10     # Wejście do pętli głównej tkinter.
11     tkinter.mainloop()
12
13 # Utworzenie egzemplarza klasy MyGUI.
14 my_gui = MyGUI()
```



Rysunek 13.4. Okno wyświetlone przez program z listingu 13.1

W wierszach od 5. do 11. znajduje się definicja klasy `MyGUI`. Metoda `__init__()` tej klasy zaczyna się w wierszu 6. Polecenie w wierszu 8. tworzy widżet główny i przypisuje go zmiennej `main_window`. W wierszu 11. mamy wywołanie funkcji `mainloop()` modułu `tkinter`. Polecenie w wierszu 14. tworzy egzemplarz klasy `MyGUI`. To powoduje wykonanie metody `__init__()` klasy i wyświetlenie pustego okna na ekranie.

Punkt kontrolny

13.5. Pokrótce wyjaśnij przeznaczenie następujących widżetów modułu `tkinter`:
   a. `Label`,
   b. `Entry`,
   c. `Button`,
   d. `Frame`.

13.6. Jak można utworzyć widżet główny?

13.7. Do czego służy metoda `mainloop()` modułu `tkinter`?

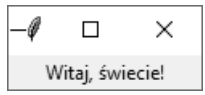
13.3. Wyświetlanie tekstu za pomocą widżetu Label

WYJAŚNIENIE. Widżet `Label` można wykorzystać do wyświetlenia tekstu w oknie.

Widżet `Label` można wykorzystać do wyświetlenia jednego wiersza tekstu w oknie. Widżet wymaga utworzenia egzemplarza klasy `Label` modułu `tkinter`. Program przedstawiony na listingu 13.3 tworzy okno wraz z widżetem `Label` wyświetlającym komunikat `Witaj, świecie!`. Okno pokazałem na rysunku 13.5.

Listing 13.3 (hello_world.py)

```
1  # Ten program wyświetla etykietę wraz z tekstem.
2
3  import tkinter
4
5  class MyGUI:
6    def __init__(self):
7        # Utworzenie widżetu okna głównego.
8        self.main_window = tkinter.Tk()
9
10       # Utworzenie widżetu Label zawierającego
11       # komunikat 'Witaj, świecie!'
12       self.label = tkinter.Label(self.main_window,
13                                  text='Witaj, świecie!')
14
15       # Wywołanie metody pack() widżetu Label.
16       self.label.pack()
17
18       # Wejście do pętli głównej tkinter.
19       tkinter.mainloop()
20
21 # Utworzenie egzemplarza klasy MyGUI.
22 my_gui = MyGUI()
```



Rysunek 13.5. Okno wyświetlone przez program z listingu 13.3

Klasa `MyGUI` jest bardzo podobna do użytej w programie z listingu 13.2. Jej metoda `__init__()` generuje graficzny interfejs użytkownika podczas tworzenia egzemplarza klasy. W wierszu 8. mamy utworzenie widżetu głównego i przypisanie go `self.main_window`. W wierszach 12. i 13. znajduje się polecenie:

```
self.label = tkinter.Label(self.main_window,
                           text='Witaj, świecie!')
```

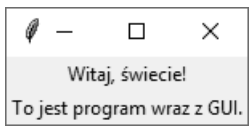
Tworzy ono widżet `Label` i przypisuje go `self.label`. Pierwszym argumentem jest `self.main_window`, czyli odwołanie do widżetu głównego. To po prostu oznacza, że widżet `Label` ma należeć do widżetu głównego. Drugim argumentem jest `text='Witaj, świecie!'`. To z kolei określa tekst, który ma zostać wyświetlony w etykiecie.

Polecenie w wierszu 16. wywołuje metodę `pack()` widżetu `Label`. Jej działanie polega na ustaleniu miejsca umieszczenia widżetu, a następnie wyświetlenia go po utworzeniu okna głównego. (Tę metodę można wywołać dla każdego widżetu w oknie). Wiersz 19. zawiera wywołanie metody `mainloop()` modułu `tkinter`, która wyświetla okno główne programu pokazane na rysunku 13.5.

Spójrz na kolejny przykład. Program przedstawiony na listingu 13.4 wyświetla okno wraz z dwoma widżetami `Label`, co pokazałem na rysunku 13.6.

Listing 13.4 (hello_world2.py)

```
1  # Ten program wyświetla dwie etykiety wraz z tekstem.
2
3  import tkinter
4
5  class MyGUI:
6      def __init__(self):
7          # Utworzenie widżetu okna głównego.
8          self.main_window = tkinter.Tk()
9
10         # Utworzenie dwóch widżetów Label.
11         self.label1 = tkinter.Label(self.main_window,
12                                     text='Witaj, świecie!')
13         self.label2 = tkinter.Label(self.main_window,
14                                     text='To jest program wraz z GUI.')
15
16         # Wywołanie metody pack() w obu widżetach Label.
17         self.label1.pack()
18         self.label2.pack()
19
20         # Wejście do pętli głównej tkinter.
21         tkinter.mainloop()
22
23 # Utworzenie egzemplarza klasy MyGUI.
24 my_gui = MyGUI()
```

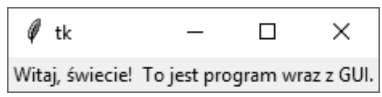


Rysunek 13.6. Okno wyświetlone przez program z listingu 13.4

Zwróć uwagę na wyświetlenie dwóch widżetów `Label`, jeden po drugim. To ułożenie można zmienić za pomocą argumentu metody `pack()`. Przykład jego użycia pokazałem w programie na listingu 13.5. Po jego uruchomieniu zostanie wyświetlone okno, takie jak na rysunku 13.7.

Listing 13.5 (hello_world3.py)

```
1  # Ten program używa argumentu side='left' metody
2  # pack(), aby zmienić układ ułożenia etykiet.
3
4  import tkinter
5
6  class MyGUI:
7      def __init__(self):
8          # Utworzenie widżetu okna głównego.
9          self.main_window = tkinter.Tk()
10
11         # Utworzenie dwóch widżetów Label.
12         self.label1 = tkinter.Label(self.main_window,
13                                     text='Witaj, świecie!')
14         self.label2 = tkinter.Label(self.main_window,
15                                     text='To jest program wraz z GUI.')
16
17         # Wywołanie metody pack() w obu widżetach Label.
18         self.label1.pack(side='left')
19         self.label2.pack(side='left')
20
21         # Wejście do pętli głównej tkinter.
22         tkinter.mainloop()
23
24   # Utworzenie egzemplarza klasy MyGUI.
25 my_gui = MyGUI()
```



Rysunek 13.7. Okno wyświetlone przez program z listingu 13.5

W wierszach 18. i 19. znajdują się wywołania metod `pack()` obu widżetów `Label` wraz z argumentami `side='Left'`. Ten argument określa, że widżet ma zostać dosunięty do lewej strony w widżecie nadrzędnym. Ponieważ widżet `label1` został jako pierwszy dodany do `main_window`, będzie wyświetlony przy lewej krawędzi okna. Natomiast widżet `label2` został dodany jako drugi i dlatego jest wyświetlony po `label1`. W efekcie etykiety zostały wyświetlone obok siebie. Poprawne argumenty możliwe do przekazania metodzie `pack()` to `side='top'`, `side='bottom'`, `side='left'` i `side='right'`.

Punkt kontrolny

13.8. Na czym polega działanie metody `pack()` widżetu?

13.9. Przyjmuję założenie o utworzeniu dwóch widżetów `Label` i wywołaniu ich metod `pack()` bez argumentów. Jak zostaną rozmieszczone te widżety?

13.10. Jaki argument trzeba przekazać metodzie `pack()` widżetu, aby nakazać jego wyrównanie do lewej strony wewnątrz widżetu nadrzędnego?

13.4. Organizowanie widżetów przy użyciu kontenera Frame

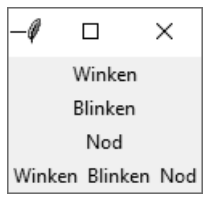
WYJAŚNIENIE. **`Frame` to kontener przechowujący widżety. Kontenery te można wykorzystać do organizowania widżetów w oknie.**

Kontener `Frame` to rodzaj widżetu, który może przechowywać inne widżety. Kontener ten przydaje się do organizowania i aranżowania grup widżetów w oknie. Przykładowo zbiór widżetów można umieścić w jednym kontenerze `Frame` i rozmieścić w dowolny sposób, a następnie umieścić kolejny zbiór widżetów w innym kontenerze `Frame` i rozmieścić je w odmienny sposób. Takie rozwiązanie pokazałem w programie na listingu 13.6. Po jego uruchomieniu zobaczysz okno pokazane na rysunku 13.8.

Listing 13.6 (frame_demo.py)

```
# Ten program tworzy etykiety w dwóch oddzielnych kontenerach.

import tkinter

class MyGUI:
    def __init__(self):
        # Utworzenie widżetu okna głównego.
        self.main_window = tkinter.Tk()

        # Utworzenie dwóch kontenerów w oknie,
        # pierwszy na górze, a drugi pod nim.
        self.top_frame = tkinter.Frame(self.main_window)
        self.bottom_frame = tkinter.Frame(self.main_window)

        # Utworzenie trzech widżetów Label
        # w górnym kontenerze.
        self.label1 = tkinter.Label(self.top_frame,
                                    text='Winken')
        self.label2 = tkinter.Label(self.top_frame,
                                    text='Blinken')
        self.label3 = tkinter.Label(self.top_frame,
                                    text='Nod')

        # Ułożenie etykiet w górnym kontenerze.
        # Za pomocą argumentu side='top' zostały
        # rozmieszczone pionowo jedna po drugiej.
        self.label1.pack(side='top')
        self.label2.pack(side='top')
        self.label3.pack(side='top')

        # Utworzenie trzech widżetów Label
        # w dolnym kontenerze.
        self.label4 = tkinter.Label(self.bottom_frame,
                                    text='Winken')
        self.label5 = tkinter.Label(self.bottom_frame,
                                    text='Blinken')
        self.label6 = tkinter.Label(self.bottom_frame,
                                    text='Nod')
```

```
39
40          # Ułożenie etykiet w dolnym kontenerze.
41          # Za pomocą argumentu side='left' zostały
42          # rozmieszczone poziomo jedna po drugiej.
43          self.label4.pack(side='left')
44          self.label5.pack(side='left')
45          self.label6.pack(side='left')
46
47          # Tak, w kontenerze również konieczne jest wywołanie metody pack()!
48          self.top_frame.pack()
49          self.bottom_frame.pack()
50
51          # Wejście do pętli głównej tkinter.
52          tkinter.mainloop()
53
54  # Utworzenie egzemplarza klasy MyGUI.
55  my_gui = MyGUI()
```



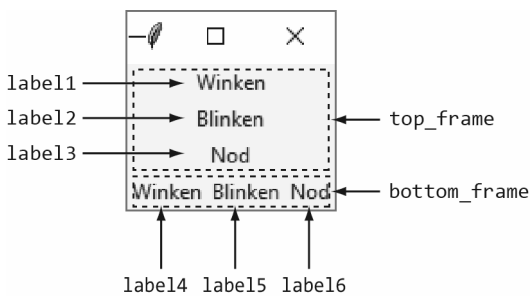
Rysunek 13.8. Okno wyświetlone przez program z listingu 13.6

Przyjrzyj się dokładnie poleceniom w wierszach 12. i 13.

```
self.top_frame = tkinter.Frame(self.main_window)
self.bottom_frame = tkinter.Frame(self.main_window)
```

Polecenia te powodują utworzenie dwóch obiektów `Frame`. Argument `self.main_window` powoduje dodanie tych obiektów do widżetu `main_window`.

W wierszach od 17. do 22. następuje utworzenie trzech widżetów `Label`. Zwróć uwagę na ich dodanie do widżetu `self.top_frame`. Następnie w wierszach od 27. do 29. mamy wywołanie metody `pack()` poszczególnych widżetów `Label` wraz z argumentem `side='top'`. Jak pokazałem na rysunku 13.9, powoduje to pionowe rozmieszczenie etykiet, jedna po drugiej, wewnątrz górnego kontenera `Frame`.



Rysunek 13.9. Rozmieszczenie widżetów w programie z listingu 13.6

W wierszach od 33. do 38. następuje utworzenie trzech kolejnych widżetów `Label`. Zwróć uwagę na ich dodanie do widżetu `self.bottom_frame`. Następnie w wierszach od 43. do 45. mamy wywołanie metody `pack()` nowych widżetów `Label` wraz z argumentem `side='left'`. Jak pokazałem na rysunku 13.9, powoduje to poziome rozmieszczenie etykiet, jedna po drugiej, wewnątrz dolnego kontenera `Frame`.

W wierszach 48. i 49. następuje wywołanie metody `pack()` w obu kontenerach, co powoduje wyświetlenie widżetów `Frame`. Z kolei w wierszu 52. mamy wywołanie funkcji `mainloop()` modułu `tkinter`.

13.5. Widżet Button i informacyjne okna dialogowe

WYJAŚNIENIE. **Widżetu `Button` można użyć do utworzenia standardowego przycisku w oknie. Kliknięcie takiego przycisku przez użytkownika spowoduje wywołanie określonej funkcji lub metody.**

Informacyjne okno dialogowe to proste okno wyświetlające użytkownikowi komunikat wraz z przyciskiem *OK* pozwalającym na zamknięcie tego okna. Do wyświetlenia informacyjnego okna dialogowego można użyć funkcji `showinfo()` modułu `tkinter.messagebox`.

Widżet `Button` może być kliknięty przez użytkownika i powoduje wówczas przeprowadzenie pewnej operacji. Podczas tworzenia widżetu `Button` można zdefiniować tekst wyświetlany w przycisku i nazwę funkcji wywołania zwrotnego. **Funkcja wywołania zwrotnego** to funkcja lub metoda wywoływana po kliknięciu przycisku przez użytkownika.

UWAGA. Funkcja wywołania zwrotnego jest określana także mianem **procedury obsługi**, ponieważ obsługuje zdarzenie występujące po kliknięciu przycisku.

Spójrz na program przedstawiony na listingu 13.7. Jego działanie polega na wyświetleniu okna, które możesz zobaczyć na rysunku 13.10. Gdy użytkownik kliknie przycisk, program wyświetli oddzielne **informacyjne okno dialogowe** pokazane na rysunku 13.11. Do wyświetlenia tego okna wykorzystałem funkcję o nazwie `showinfo()` modułu `tkinter.messagebox()`. Aby użyć wymienionej funkcji, trzeba zaimportować moduł `tkinter.messagebox`. Oto ogólna postać wywołania funkcji `showinfo()`.

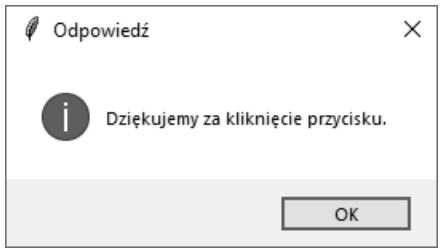
```
tkinter.messagebox.showinfo(tytuł, komunikat)
```

W tej ogólnej postaci *`tytuł`* to ciąg tekstowy wyświetlany w pasku tytułu okna dialogowego, natomiast *`komunikat`* to informacyjny ciąg tekstowy wyświetlony w części głównej okna dialogowego.

Listing 13.7 (button_demo.py)

```
# Ten program pokazuje widżet Button.
# Po jego kliknięciu przez użytkownika
# zostanie wyświetlone informacyjne okno dialogowe.

import tkinter
import tkinter.messagebox

class MyGUI:
    def __init__(self):
        # Utworzenie widżetu okna głównego.
        self.main_window = tkinter.Tk()

        # Utworzenie widżetu Button. Tekst 'Kliknij mnie!'
        # powinien zostać wyświetlony na przycisku. Metoda
        # do_something() będzie wywołana, gdy użytkownik
        # kliknie ten widżet.
        self.my_button = tkinter.Button(self.main_window,
                                        text='Kliknij mnie!',
                                        command=self.do_something)

        # Wywołanie metody pack() widżetu.
        self.my_button.pack()

        # Wejście do pętli głównej tkinter.
        tkinter.mainloop()

  # Metoda do_something() to funkcja wywołania
  # zwrotnego dla widżetu Button.

  def do_something(self):
        # Wyświetlenie informacyjnego okna dialogowego.
        tkinter.messagebox.showinfo('Odpowiedź',
                                    'Dziękujemy za kliknięcie przycisku.')

# Utworzenie egzemplarza klasy MyGUI.
my_gui = MyGUI()
```

Rysunek 13.10. Okno główne wyświetlone przez program z listingu 13.7



Rysunek 13.11. Informacyjne okno dialogowe wyświetlone przez program z listingu 13.7

W wierszu 5. został zaimportowany moduł `tkinter`, natomiast w wierszu 6. `tkinter.messagebox`. Z kolei w wierszu 11. mamy utworzenie widżetu głównego i przypisanie go zmiennej `main_window`.

Polecenia w wierszach od 17. do 19. tworzą widżet `Button`. Pierwszym argumentem jest `self.main_window`, czyli widżet nadrzędny. Drugi argument, `text='Kliknij mnie!'`, określa ciąg tekstowy, który ma zostać wyświetlony na przycisku. Trzeci argument, `command='self.do_something'`, wskazuje metodę `do_something()` klasy jako funkcję wywołania zwrotnego. Po kliknięciu przycisku przez użytkownika nastąpi wywołanie wymienionej funkcji.

Metoda `do_something()` została zdefiniowana w wierszach od 31. do 33. Jej działanie polega na wywołaniu funkcji `tkinter.messagebox.showinfo()` i wyświetleniu informacyjnego okna dialogowego pokazanego na rysunku 13.11. Aby je zamknąć, należy kliknąć przycisk *OK*.

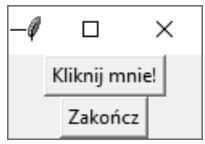
Utworzenie przycisku zakończenia działania programu

Programy wyposażone w graficzny interfejs użytkownika zwykle mają przycisk **kończący działanie programu**, którego kliknięcie powoduje zamknięcie aplikacji. Aby utworzyć taki przycisk w Pythonie, wystarczy zdefiniować widżet `Button` wywołujący jako funkcję wywołania zwrotnego metodę `destroy()` widżetu głównego. Program przedstawiony na listingu 13.8 pokazuje przykład takiego rozwiązania. Jest to zmodyfikowana wersja wcześniejszego programu, w której dodałem drugi widżet `Button`. Uruchomiony program możesz zobaczyć na rysunku 13.12.

Listing 13.8 (quit_button.py)

```
1  # Ten program ma przycisk kończący działanie aplikacji.
2  # Po kliknięciu przycisk wywołuje metodę destroy() klasy Tk.
3
4  import tkinter
5  import tkinter.messagebox
6
7  class MyGUI:
8      def __init__(self):
9          # Utworzenie widżetu okna głównego.
10         self.main_window = tkinter.Tk()
11
12         # Utworzenie widżetu Button. Tekst 'Kliknij mnie!'
13         # powinien zostać wyświetlony na przycisku. Metoda
14         # do_something() będzie wywołana, gdy użytkownik
15         # kliknie ten widżet.
16         self.my_button = tkinter.Button(self.main_window,
17                                         text='Kliknij mnie!',
18                                         command=self.do_something)
19
20         # Utworzenie przycisku zamykającego program. Po jego kliknięciu
21         # nastąpi wywołanie metody destroy() widżetu głównego.
22         # (Zmienna main_window odwołuje się do widżetu głównego,
23         # więc funkcją wywołania zwrotnego jest self.main_window.destroy()).
24         self.quit_button = tkinter.Button(self.main_window,
```

```
25                                          text='Zakończ',
26                                          command=self.main_window.destroy)
27
28
29          # Wywołanie metody pack() widżetu.
30          self.my_button.pack()
31          self.quit_button.pack()
32
33          # Wejście do pętli głównej tkinter.
34          tkinter.mainloop()
35
36      # Metoda do_something() to funkcja wywołania
37      # zwrotnego dla widżetu Button.
38
39      def do_something(self):
40          # Wyświetlenie informacyjnego okna dialogowego.
41          tkinter.messagebox.showinfo('Odpowiedź',
42                                      'Dziękujemy za kliknięcie przycisku.')
43
44  # Utworzenie egzemplarza klasy MyGUI.
45  my_gui = MyGUI()
```



Rysunek 13.12. Informacyjne okno dialogowe wyświetlone przez program z listingu 13.8

Polecenia w wierszach od 24. do 26. powodują utworzenie przycisku kończącego działanie programu. Zwróć uwagę na użycie metody `self.main_window.destroy()` w charakterze funkcji wywołania zwrotnego. Po kliknięciu tego przycisku zostaje wywołana wymieniona metoda i program kończy działanie.

13.6. Pobieranie danych wejściowych za pomocą widżetu Entry

WYJAŚNIENIE. **Widżet `Entry` to prostokątny obszar, w którym użytkownik może wprowadzić dane wejściowe. Metodę `get()` tego widżetu można wykorzystać do otrzymania wpisanych danych.**

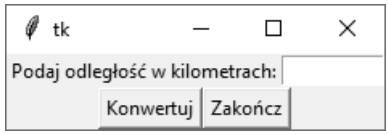
Widżet `Entry` to prostokątny obszar, w którym użytkownik może wpisać tekst. Widżet jest używany do pobierania danych wejściowych w programach wyposażonych w GUI. Zwykle program będzie miał jeden lub więcej widżetów `Entry` w oknie i przycisk, którego kliknięcie spowoduje wysłanie wprowadzonych danych. Funkcja wywołania zwrotnego przycisku otrzymuje dane z okna zawierającego widżet `Entry` i przetwarza je.

Metodę `get()` widżetu `Entry` można wykorzystać do pobrania danych wprowadzonych przez użytkownika za pomocą tego widżetu. Wartością zwrotną metody `get()` jest ciąg tekstowy, więc dane muszą być skonwertowane na odpowiedni typ, jeśli widżet jest używany do pobrania danych liczbowych.

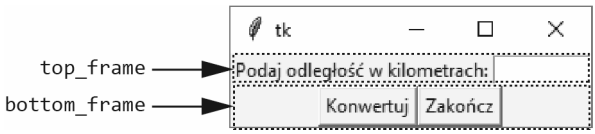
Aby pokazać w praktyce omawiany widżet, przedstawię program pozwalający użytkownikowi na wpisanie wartości wyrażonej w kilometrach, która po kliknięciu przycisku zostanie skonwertowana na wartość w milach. Oto wzór pozwalający skonwertować kilometry na mile:

$$mile = kilometry \times 0{,}6214$$

Na rysunku 13.13 pokazałem okno wyświetlane przez omawiany program. Aby ułożyć widżety w sposób pokazany na rysunku, należy umieścić je w dwóch kontenerach `Frame`, tak jak pokazałem na rysunku 13.14. Etykieta i widżet `Entry` umieszczane są w kontenerze `top_frame`, a ich metody `pack()` będą wywoływane wraz z argumentem `side='left'`. To spowoduje poziome ułożenie wymienionych elementów. Z kolei przyciski *Konwertuj* i *Zakończ* zostaną umieszczone w kontenerze `bottom_frame`, a ich metody `pack()` również będą wywoływane wraz z argumentem `side='left'`.



Rysunek 13.13. Okno programu kilo_converter.py



Rysunek 13.14. Ułożenie widżetów za pomocą kontenerów

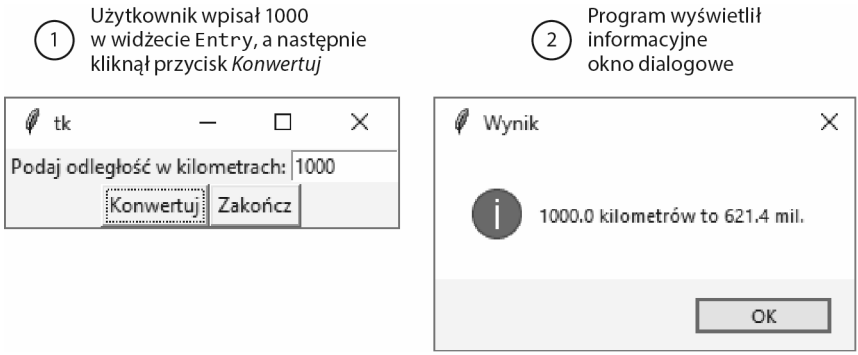
Kod omawianego programu przedstawiłem na listingu 13.9. Na rysunku 13.15 możesz zobaczyć, co się stanie, gdy użytkownik wpisze w widżecie `Entry` liczbę `1000` i kliknie przycisk *Konwertuj*.

Listing 13.9 (kilo_converter.py)

```
1  # Ten program konwertuje odległość wyrażoną w kilometrach
2  # na mile. Wynik zostanie wyświetlony
3  # w informacyjnym oknie dialogowym.
4
5  import tkinter
6  import tkinter.messagebox
7
8  class KiloConverterGUI:
9      def __init__(self):
10
11         # Utworzenie okna głównego.
12         self.main_window = tkinter.Tk()
```

```
13
14          # Utworzenie dwóch kontenerów do grupowania widżetów.
15          self.top_frame = tkinter.Frame(self.main_window)
16          self.bottom_frame = tkinter.Frame(self.main_window)
17
18          # Utworzenie widżetów w górnym kontenerze.
19          self.prompt_label = tkinter.Label(self.top_frame,
20                                            text='Podaj odległość w kilometrach:')
21          self.kilo_entry = tkinter.Entry(self.top_frame,
22                                          width=10)
23
24          # Wywołanie metod pack() widżetów w górnym kontenerze.
25          self.prompt_label.pack(side='left')
26          self.kilo_entry.pack(side='left')
27
28          # Utworzenie widżetów Button w dolnym kontenerze.
29          self.calc_button = tkinter.Button(self.bottom_frame,
30                                            text='Konwertuj',
31                                            command=self.convert)
32          self.quit_button = tkinter.Button(self.bottom_frame,
33                                            text='Zakończ',
34                                            command=self.main_window.destroy)
35          # Wywołanie metody pack() przycisków.
36          self.calc_button.pack(side='left')
37          self.quit_button.pack(side='left')
38
39          # Wywołanie metody pack() kontenerów.
40          self.top_frame.pack()
41          self.bottom_frame.pack()
42
43          # Wejście do pętli głównej tkinter.
44          tkinter.mainloop()
45
46      # Metoda convert() jest funkcją wywołania
47      # zwrotnego przycisku Konwertuj.
48
49      def convert(self):
50          # Pobranie wartości wprowadzonej
51          # przez użytkownika w widżecie kilo_entry.
52          kilo = float(self.kilo_entry.get())
53
54          # Konwersja kilometrów na mile.
55          miles = kilo * 0.6214
56
57          # Wyświetlenie wyniku w informacyjnym oknie dialogowym.
58          tkinter.messagebox.showinfo('Wynik',
59                                      str(kilo) +
60                                      ' kilometrów to ' +
61                                      str(miles) + ' mil.')
62
63  # Utworzenie egzemplarza klasy KiloConverterGUI.
64  kilo_conv = KiloConverterGUI()
```

Zdefiniowana w wierszach od 49. do 60. metoda `convert()` jest funkcją wywołania zwrotnego dla przycisku *Konwertuj*. Polecenie w wierszu 52. wywołuje metodę `get()` widżetu `kilo_entry` w celu pobrania danych wpisanych przez użytkownika. Dane zostają skonwertowane na wartość typu `float` i przypisane zmiennej `kilo`. Obliczenie



Rysunek 13.15. Informacyjne okno dialogowe wyświetlone przez program z listingu 13.9

mil odbywa się w wierszu 55., a wynik jest przypisywany zmiennej `miles`. Następnie polecenia w wierszach od 58. do 61. wyświetlają informacyjne okno dialogowe wraz ze skonwertowaną wartością.

13.7. Używanie etykiety jako elementu danych wyjściowych

WYJAŚNIENIE. **Gdy obiekt `StringVal` jest powiązany z widżetem `Label`, widżet ten wyświetla wszystkie dane przechowywane w wymienionym obiekcie.**

W poprzednim przykładzie zobaczyłeś, jak można użyć informacyjnego okna dialogowego do wyświetlenia danych wyjściowych. Jeżeli nie chcesz wyświetlać oddzielnego okna dialogowego dla danych wyjściowych programu, zamiast niego możesz użyć widżetu `Label` w oknie głównym programu. Pozwala to na dynamiczne wyświetlanie danych wyjściowych programu. Wystarczy utworzyć pusty widżet `Label` w oknie głównym, a następnie przygotować kod wyświetlający w tej etykiecie żądane dane po kliknięciu przycisku.

Moduł `tkinter` oferuje klasę o nazwie `StringVal`, którą wraz z widżetem `Label` można wykorzystać do wyświetlania danych. Pierwszym krokiem jest utworzenie obiektu `StringVar`. Następnie trzeba opracować widżet `Label` i powiązać go z obiektem `StringVal`. Od tej chwili każda wartość przechowywana w obiekcie `StringVal` będzie automatycznie wyświetlana przez widżet `Label`.

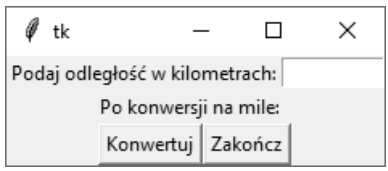
Zastosowanie takiego rozwiązania w praktyce pokazałem w programie na listingu 13.10. Jest to zmodyfikowana wersja programu *kilo_converter.py* z listingu 13.9. Zamiast wyświetlać wynik w informacyjnym oknie dialogowym, ta wersja programu wyświetla wynik w etykiecie znajdującej się w oknie głównym.

Listing 13.10 (kilo_converter2.py)

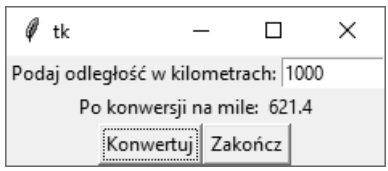
```
# Ten program konwertuje odległość wyrażoną w kilometrach
# na mile. Wynik zostanie
# wyświetlony w etykiecie okna głównego.

import tkinter

class KiloConverterGUI:
    def __init__(self):

        # Utworzenie okna głównego.
        self.main_window = tkinter.Tk()

        # Utworzenie trzech kontenerów do grupowania widżetów.
        self.top_frame = tkinter.Frame()
        self.mid_frame = tkinter.Frame()
        self.bottom_frame = tkinter.Frame()

        # Utworzenie widżetów w górnym kontenerze.
        self.prompt_label = tkinter.Label(self.top_frame,
                                          text='Podaj odległość w kilometrach:')
        self.kilo_entry = tkinter.Entry(self.top_frame,
                                        width=10)

        # Wywołanie metod pack() widżetów w górnym kontenerze.
        self.prompt_label.pack(side='left')
        self.kilo_entry.pack(side='left')

        # Utworzenie widżetów w środkowym kontenerze.
        self.descr_label = tkinter.Label(self.mid_frame,
                                         text='Po konwersji na mile:')

        # Konieczne jest powiązanie obiektu StringVar z etykietą
        # danych wyjściowych. Metoda set() obiektu będzie
        # użyta do przechowywania pustego ciągu tekstowego.
        self.value = tkinter.StringVar()

        # Utworzenie etykiety i powiązanie jej
        # z obiektem StringVar. Każda wartość
        # przechowywana w tym obiekcie zostanie
        # automatycznie wyświetlona w etykiecie.
        self.miles_label = tkinter.Label(self.mid_frame,
                                         textvariable=self.value)

        # Wywołanie metod pack() widżetów w środkowym kontenerze.
        self.descr_label.pack(side='left')
        self.miles_label.pack(side='left')

        # Utworzenie widżetów Button w dolnym kontenerze.
        self.calc_button = tkinter.Button(self.bottom_frame,
                                          text='Konwertuj',
                                          command=self.convert)
        self.quit_button = tkinter.Button(self.bottom_frame,
                                          text='Zakończ',
                                          command=self.main_window.destroy)

        # Wywołanie metody pack() przycisków.
        self.calc_button.pack(side='left')
        self.quit_button.pack(side='left')
```

```
59
60          # Wywołanie metody pack() kontenerów.
61          self.top_frame.pack()
62          self.mid_frame.pack()
63          self.bottom_frame.pack()
64
65          # Wejście do pętli głównej tkinter.
66          tkinter.mainloop()
67
68      # Metoda convert() jest funkcją wywołania
69      # zwrotnego przycisku Konwertuj.
70
71      def convert(self):
72          # Pobranie wartości wprowadzonej
73          # przez użytkownika w widżecie kilo_entry.
74          kilo = float(self.kilo_entry.get())
75
76          # Konwersja kilometrów na mile.
77          miles = kilo * 0.6214
78
79          # Konwersja wartości miles na postać ciągu tekstowego
80          # i umieszczenie jej w obiekcie StringVal. To spowoduje
81          # automatyczne uaktualnienie widżetu miles_label.
82          self.value.set(miles)
83
84  # Utworzenie egzemplarza klasy KiloConverterGUI.
85  kilo_conv = KiloConverterGUI()
```

Po uruchomieniu tego programu zostanie wyświetlone okno pokazane na rysunku 13.16. Z kolei na rysunku 13.17 pokazałem, co się stanie, gdy użytkownik wpisze w widżecie `Entry` liczbę `1000` i kliknie przycisk *Konwertuj*. Skonwertowana wartość w milach zostanie wyświetlona w etykiecie w oknie głównym.



Rysunek 13.16. Okno wyświetlone po uruchomieniu programu

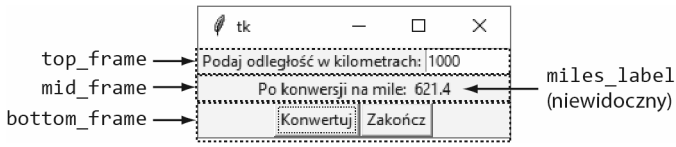


Rysunek 13.17. Okno pokazujące wartość 1000 kilometrów skonwertowaną na mile

Spójrz na kod źródłowy omawianego programu. W wierszach od 14. do 16. następuje utworzenie trzech kontenerów: `top_frame`, `mid_frame` i `bottom_frame`. Następnie w wierszach od 19. do 26. tworzone są widżety dla górnego kontenera i zostają wywołane ich metody `pack()`.

W wierszach 29. i 30. znajduje się polecenie tworzące widżet `Label` wraz z tekstem `Po konwersji na mile`, który możesz zobaczyć w oknie głównym na rysunku 13.16. Następnie w wierszu 35. zostaje utworzony obiekt `StringVal` przypisywany zmiennej `value`. W wierszu 41. jest opracowywany widżet `Label` o nazwie `miles_label` przeznaczony do wyświetlania wartości skonwertowanej na mile. Zwróć uwagę na użycie w wierszu 42. argumentu `textvariable=self.value`. Powoduje on powstanie powiązania między widżetem `Label` i obiektem `StringVar`, do którego odwołuje się zmienna `value`. Każda wartość przechowywana przez obiekt `StringVal` zostanie wyświetlona w etykiecie.

W wierszach 45. i 46. następuje wywołanie metod `pack()` widżetów `Label` umieszczonych w kontenerze `mid_frame`. Z kolei w wierszach od 49. do 58. są tworzone widżety `Button` i wywoływane ich metody `pack()`. Następnie w wierszach od 61. do 63. mamy wywołanie metod `pack()` kontenerów. Na rysunku 13.18 pokazałem ułożenie różnych widżetów w trzech kontenerach w oknie omawianego programu.



Rysunek 13.18. Układ okna głównego programu kilo_converter2.py

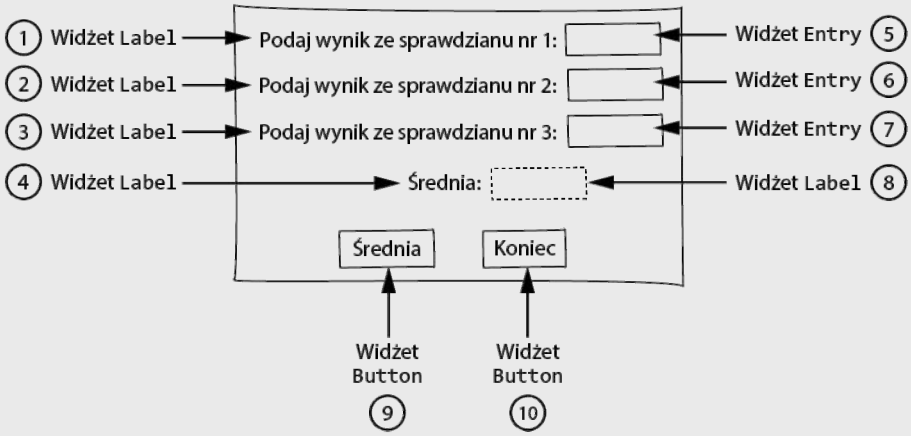
Zdefiniowana w wierszach od 71. do 82. metoda `convert()` jest funkcją wywołania zwrotnego dla przycisku *Konwertuj*. Polecenie w wierszu 74. wywołuje metodę `get()` widżetu `kilo_entry`, aby pobrać dane wprowadzone przez użytkownika. Zostają one skonwertowane na wartość typu `float` przypisywaną później zmiennej `kilo`. Obliczenia w wierszu 77. przeprowadzają konwersję i przypisują jej wynik zmiennej `miles`. Polecenie w wierszu 82. wywołuje metodę `set()` obiektu `StringVar` wraz z argumentem `miles`. Powoduje to umieszczenie w obiekcie `StringVal` wartości wskazywanej przez zmienną `miles` i wyświetlenie tej wartości przez widżet `miles_label`.

W centrum uwagi

Utworzenie programu z GUI

Kasia uczy w szkole biologii. W rozdziale 3. prześledziliśmy proces tworzenia programu, za pomocą którego uczniowie Kasi mogą obliczać średni wynik z trzech sprawdzianów. Program prosił ucznia o podanie wyniku z każdego sprawdzianu, a następnie wyświetlał średni wynik. Kasia poprosiła Cię, abyś tym razem opracował program wyposażony w GUI, który będzie wykonywał tę samą operację. Kasia chce, aby w oknie programu znajdowały się trzy pola tekstowe, w których będzie można wpisać wyniki ze sprawdzianów, i przycisk, po którego kliknięciu wyświetli się średni wynik.

Na samym początku musimy naszkicować okno programu — przedstawiłem je na rysunku 13.19. Na szkicu widoczne są także typy widżetów (ponumerowanie ich ułatwi przygotowanie listy).

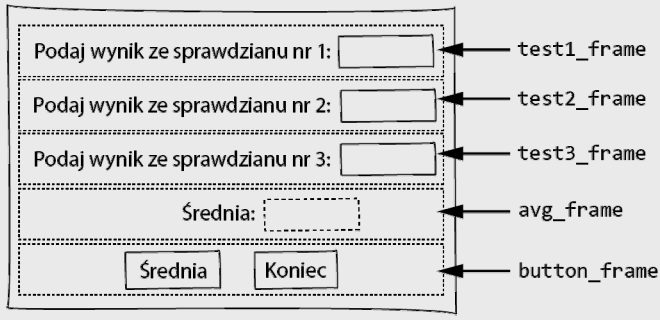


Rysunek 13.19. Szkic okna budowanego programu

Na podstawie szkicu możemy sporządzić listę potrzebnych widżetów. Opracowując przedstawioną w następującej tabeli listę, dodamy także krótki opis każdego widżetu i nazwę, jaką mu nadamy podczas tworzenia okna.

Numer widżetu na rysunkach od 13.13 do 13.19	Typ widżetu	Opis	Nazwa
1	`Label`	Prosi użytkownika o podanie wyniku ze sprawdzianu nr 1	`test1_label`
2	`Label`	Prosi użytkownika o podanie wyniku ze sprawdzianu nr 2	`test2_label`
3	`Label`	Prosi użytkownika o podanie wyniku ze sprawdzianu nr 3	`test3_label`
4	`Label`	Średnia z podanych sprawdzianów	`result_label`
5	`Entry`	W tym miejscu użytkownik podaje wynik ze sprawdzianu nr 1	`test1_entry`
6	`Entry`	W tym miejscu użytkownik podaje wynik ze sprawdzianu nr 2	`test2_entry`
7	`Entry`	W tym miejscu użytkownik podaje wynik ze sprawdzianu nr 3	`test3_entry`
8	`Label`	W tej etykiecie program wyświetli średni wynik z podanych sprawdzianów	`avg_label`
9	`Button`	Kliknięcie tego przycisku powoduje obliczenie średniego wyniku ze sprawdzianów i wyświetlenie go w etykiecie `avg_label`	`calc_button`
10	`Button`	Kliknięcie tego przycisku powoduje zakończenie działania programu	`quit_button`

Na podstawie szkicu widzisz, że program zawiera pięć wierszy widżetów. Aby je zorganizować, wykorzystamy pięć kontenerów `Frame`. Na rysunku 13.20 pokazałem sposób rozmieszczenia tych widżetów w pięciu kontenerach `Frame`.



Rysunek 13.20. Użycie kontenerów Frame do rozmieszczenia widżetów w oknie programu

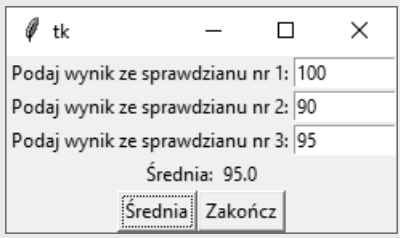
Kod źródłowy omawianego programu przedstawiłem na listingu 13.11. Z kolei na rysunku 13.21 pokazałem okno programu wraz z danymi wprowadzonymi przez użytkownika.

Listing 13.11 (test_averages.py)

```
1  # Ten program używa GUI do pobrania wyników
2  # z trzech sprawdzianów i wyświetlenia ich średniej.
3
4  import tkinter
5
6  class TestAvg:
7      def __init__(self):
8          # Utworzenie okna głównego.
9          self.main_window = tkinter.Tk()
10
11         # Utworzenie pięciu kontenerów.
12         self.test1_frame = tkinter.Frame(self.main_window)
13         self.test2_frame = tkinter.Frame(self.main_window)
14         self.test3_frame = tkinter.Frame(self.main_window)
15         self.avg_frame = tkinter.Frame(self.main_window)
16         self.button_frame = tkinter.Frame(self.main_window)
17
18         # Utworzenie widżetów dla kontenera sprawdzianu nr 1 i wywołanie ich metod pack().
19         self.test1_label = tkinter.Label(self.test1_frame,
20                                          text='Podaj wynik ze sprawdzianu nr 1:')
21         self.test1_entry = tkinter.Entry(self.test1_frame,
22                                          width=10)
23         self.test1_label.pack(side='left')
24         self.test1_entry.pack(side='left')
25
26         # Utworzenie widżetów dla kontenera sprawdzianu nr 2 i wywołanie ich metod pack().
27         self.test2_label = tkinter.Label(self.test2_frame,
28                                          text='Podaj wynik ze sprawdzianu nr 2:')
29         self.test2_entry = tkinter.Entry(self.test2_frame,
30                                          width=10)
```

```
        self.test2_label.pack(side='left')
        self.test2_entry.pack(side='left')

        # Utworzenie widżetów dla kontenera sprawdzianu nr 3 i wywołanie ich metod pack().
        self.test3_label = tkinter.Label(self.test3_frame,
                                         text='Podaj wynik ze sprawdzianu nr 3:')
        self.test3_entry = tkinter.Entry(self.test3_frame,
                                         width=10)
        self.test3_label.pack(side='left')
        self.test3_entry.pack(side='left')

        # Utworzenie widżetów dla kontenera wyświetlającego średnią i wywołanie ich metod pack().
        self.result_label = tkinter.Label(self.avg_frame,
                                          text='Średnia:')
        self.avg = tkinter.StringVar() # Aby uaktualnić widżet avg_label.
        self.avg_label = tkinter.Label(self.avg_frame,
                                       textvariable=self.avg)
        self.result_label.pack(side='left')
        self.avg_label.pack(side='left')

        # Utworzenie widżetów dla kontenera przycisków i wywołanie ich metod pack().
        self.calc_button = tkinter.Button(self.button_frame,
                                          text='Średnia',
                                          command=self.calc_avg)
        self.quit_button = tkinter.Button(self.button_frame,
                                          text='Zakończ',
                                          command=self.main_window.destroy)
        self.calc_button.pack(side='left')
        self.quit_button.pack(side='left')

        # Wywołanie metody pack() kontenerów.
        self.test1_frame.pack()
        self.test2_frame.pack()
        self.test3_frame.pack()
        self.avg_frame.pack()
        self.button_frame.pack()

        # Wywołanie metody mainloop() modułu tkinter.
        tkinter.mainloop()

    # Metoda calc_avg() jest funkcją wywołania
    # zwrotnego dla widżetu calc_button.

    def calc_avg(self):
        # Pobranie wyników z trzech sprawdzianów
        # i umieszczenie ich w zmiennych.
        self.test1 = float(self.test1_entry.get())
        self.test2 = float(self.test2_entry.get())
        self.test3 = float(self.test3_entry.get())

        # Obliczenie średniego wyniku.
        self.average = (self.test1 + self.test2 +
                        self.test3) / 3.0

        # Uaktualnienie widżetu avg_label przez umieszczenie
        # wartości zmiennej self.average w obiekcie StringVar
        # wskazywanym przez avg.
        self.avg.set(self.average)
```

```
89
90  # Utworzenie egzemplarza klasy TestAvg.
91  test_avg = TestAvg()
```



Rysunek 13.21. Okno programu test_averages.py

Punkt kontrolny

13.11. Jak można pobrać dane z widżetu `Entry`?

13.12. Jaki typ ma wartość pobrana z widżetu `Entry`?

13.13. W jakim module została zdefiniowana klasa `StringVar`?

13.14. Jaki efekt można osiągnąć przez powiązanie obiektu `StringVar` z widżetem `Label`?

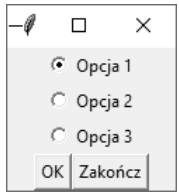
13.8. Przycisk opcji i pole wyboru

WYJAŚNIENIE. Zwykle przyciski opcji używane są w grupach składających się z co najmniej dwóch takich przycisków i pozwalają użytkownikowi na wybór jednej z wielu możliwych opcji. Z kolei pole wyboru może być stosowane samodzielnie lub w grupie i pozwala użytkownikowi na wybór typu tak-nie lub włączony-wyłączony.

Przycisk opcji

Przycisk opcji okazuje się użyteczny, gdy użytkownik ma mieć możliwość wyboru jednej z wielu dostępnych opcji. Na rysunku 13.22 pokazałem przykładowe okno zawierające grupę przycisków opcji. Taki przycisk może być zaznaczony lub nie. Każdy przycisk opcji ma małą kropkę pojawiającą się w środku, gdy dany przycisk opcji został zaznaczony. Z kolei niezaznaczony przycisk opcji wydaje się być pusty.

Do tworzenia widżetu `Radiobutton` używana jest klasa o takiej samej nazwie zdefiniowana w module `tkinter`. W danej chwili tylko jeden widżet `Radiobutton` w kontenerze,



Rysunek 13.22. Grupa przycisków opcji

takim jak `Frame`, może być zaznaczony. Kliknięcie widżetu `Radiobutton` powoduje zaznaczenie wskazanego przycisku opcji i jednocześnie automatycznie usuwa zaznaczenie z każdego innego przycisku opcji w tym samym kontenerze. Skoro tylko jeden widżet `Radiobutton` może być w danej chwili zaznaczony w kontenerze, mówimy, że te widżety są **wzajemnie wykluczające się**.

UWAGA. Nazwa przycisk opcji (ang. *radio button*) odwołuje się do starych odbiorników radiowych zawierających wciskane przyciski przeznaczone do wyboru stacji. W danej chwili mógł być wciśnięty tylko jeden przycisk. Po naciśnięciu wybranego przycisku dotychczas wybrany zostawał automatycznie wysunięty.

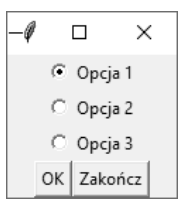
Moduł `tkinter` dostarcza klasę o nazwie `IntVar`, która może być używana wraz z widżetem `Radiobutton`. Po utworzeniu grupy przycisków opcji można je wszystkie powiązać z tym samym obiektem `IntVar`. W takim przypadku każdemu widżetowi `Radiobutton` należy przypisać unikatową wartość w postaci liczby całkowitej. Po zaznaczeniu jednego z przycisków opcji jego unikatowa wartość będzie przechowywana przez obiekt `IntVar`.

Program przedstawiony na listingu 13.12 pokazuje, jak można utworzyć widżety `Radiobutton` i jak ich używać. Na rysunku 13.23 pokazałem okno wyświetlone przez ten program. Gdy użytkownik kliknie przycisk *OK*, na ekranie zostanie wyświetlone informacyjne okno dialogowe wskazujące zaznaczony przycisk opcji.

Listing 13.12 (radiobutton_demo.py)

```
1  # Ten program pokazuje użycie grupy widżetów Radiobutton.
2  import tkinter
3  import tkinter.messagebox
4
5  class MyGUI:
6      def __init__(self):
7          # Utworzenie okna głównego.
8          self.main_window = tkinter.Tk()
9
10         # Utworzenie dwóch kontenerów, jednego dla widżetów Radiobutton,
11         # natomiast drugiego dla zwykłych widżetów Button.
12         self.top_frame = tkinter.Frame(self.main_window)
13         self.bottom_frame = tkinter.Frame(self.main_window)
14
15         # Utworzenie obiektu IntVar przeznaczonego
16         # do użycia wraz z widżetami Radiobutton.
17         self.radio_var = tkinter.IntVar()
18
19         # Przypisanie obiektowi IntVar wartości 1.
```

```
20          self.radio_var.set(1)
21
22          # Utworzenie widżetów Radiobutton w kontenerze top_frame.
23          self.rb1 = tkinter.Radiobutton(self.top_frame,
24                                         text='Opcja 1',
25                                         variable=self.radio_var,
26                                         value=1)
27          self.rb2 = tkinter.Radiobutton(self.top_frame,
28                                         text='Opcja 2',
29                                         variable=self.radio_var,
30                                         value=2)
31          self.rb3 = tkinter.Radiobutton(self.top_frame,
32                                         text='Opcja 3',
33                                         variable=self.radio_var,
34                                         value=3)
35
36          # Wywołanie metody pack() widżetów Radiobutton.
37          self.rb1.pack()
38          self.rb2.pack()
39          self.rb3.pack()
40
41          # Utworzenie przycisków OK i Zakończ.
42          self.ok_button = tkinter.Button(self.bottom_frame,
43                                          text='OK',
44                                          command=self.show_choice)
45          self.quit_button = tkinter.Button(self.bottom_frame,
46                                          text='Zakończ',
47                                          command=self.main_window.destroy)
48
49          # Wywołanie metody pack() widżetów Button.
50          self.ok_button.pack(side='left')
51          self.quit_button.pack(side='left')
52
53          # Wywołanie metody pack() kontenerów.
54          self.top_frame.pack()
55          self.bottom_frame.pack()
56
57          # Wywołanie metody mainloop() modułu tkinter.
58          tkinter.mainloop()
59
60      # Metoda show_choice() jest funkcją
61      # wywołania zwrotnego dla przycisku OK.
62      def show_choice(self):
63          tkinter.messagebox.showinfo('Wybór', 'Wybrałeś opcję ' +
64                                      str(self.radio_var.get()))
65
66  # Utworzenie egzemplarza klasy MyGUI.
67  my_gui = MyGUI()
```



Rysunek 13.23. Okno wyświetlone przez program z listingu 13.12

W wierszu 17. następuje utworzenie obiektu o nazwie `radio_var`. W wierszu 20. mamy wywołanie metody `set()` obiektu `radio_var` i umieszczenie w nim wartości `1`. (Znaczenie tej operacji poznasz już za chwilę).

W wierszach od 23. do 26. znajduje się polecenie tworzące pierwszy widżet `Radio` `↪button`. Argument `variable=self.radio_var` (wiersz 25.) powoduje powiązanie widżetu `Radiobutton` z obiektem `radio_var`. Argument `value=1` (wiersz 26.) przypisuje temu widżetowi wartość `1`. Dlatego też za każdym razem, gdy będzie zaznaczony ten przycisk opcji, obiekt `radio_var` przechowywał będzie wartość `1`.

W wierszach od 27. do 30. znajduje się polecenie tworzące drugi widżet `Radiobutton`, również powiązany z obiektem `radio_var`. Argument `value=2` (wiersz 30.) przypisuje temu widżetowi wartość `2`. Dlatego też za każdym razem, gdy będzie zaznaczony ten przycisk opcji, obiekt `radio_var` przechowywał będzie wartość `2`.

W wierszach od 31. do 34. znajduje się polecenie tworzące trzeci widżet `Radiobutton`, także powiązany z obiektem `radio_var`. Argument `value=3` (wiersz 34.) przypisuje temu widżetowi wartość `3`. Dlatego też za każdym razem, gdy będzie zaznaczony ten przycisk opcji, obiekt `radio_var` przechowywał będzie wartość `3`.

Zdefiniowana w wierszach od 62. do 64. metoda `show_choice()` jest funkcją wywołania zwrotnego dla przycisku *OK*. Wywołuje ona metodę `get()` obiektu `radio_var()` i pobiera wartość przechowywaną w obiekcie. Wartość ta zostanie następnie wyświetlona w informacyjnym oknie dialogowym.

Czy zwróciłeś uwagę na to, że po uruchomieniu programu początkowo jest zaznaczony pierwszy widżet `Radiobutton`? Tak się dzieje ze względu na przypisanie wartości `1` obiektowi `radio_var` w wierszu 20. Wymieniony obiekt może być użyty nie tylko do ustalenia zaznaczonego widżetu `Radiobutton`, ale również do wybrania określonego przycisku opcji. Gdy wartość konkretnego widżetu `Radiobutton` jest przechowywana w obiekcie `radio_var`, odpowiadający mu przycisk opcji jest zaznaczony.

Używanie funkcji wywołania zwrotnego widżetu Radiobutton

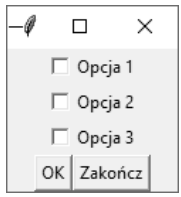
Program przedstawiony na listingu 13.12 czeka na kliknięcie przycisku *OK* przez użytkownika i dopiero wtedy ustala, który przycisk opcji został zaznaczony. Masz możliwość zdefiniowania także funkcji wywołania zwrotnego dla widżetu `Radiobutton`. Spójrz na kolejny fragment kodu.

```
self.rb1 = tkinter.Radiobutton(self.top_frame,
                               text='Opcja 1',
                               variable=self.radio_var,
                               value=1,
                               command=self.my_method)
```

Ten kod używa argumentu `command=self.my_method` do wskazania, że `my_method()` to funkcja wywołania zwrotnego. Metoda ta zostanie wywołana natychmiast po zaznaczeniu danego przycisku opcji.

Pole wyboru

Pole wyboru to małe pole wraz z etykietą wyświetloną obok niego. Na rysunku 13.24 pokazałem okno zawierające przykładową grupę trzech pól wyboru.



Rysunek 13.24. Okno zawierające przykładową grupę pól wyboru

Podobnie jak przycisk opcji, także pole wyboru może być zaznaczone lub niezaznaczone. Gdy pole opcji jest zaznaczone, będzie w nim wyświetlony mały znak przypominający literę *v*. Wprawdzie pola wyboru są często wyświetlane w grupach, ale nie są używane do dokonywania wzajemnie wykluczających się wyborów. Zamiast tego użytkownik może zaznaczyć dowolnie wybrane pole lub nie zaznaczać żadnego pola wyboru w grupie.

Do tworzenia widżetu `Checkbutton` używana jest klasa o takiej samej nazwie zdefiniowana w module `tkinter`. Podobnie jak w widżecie `Radiobutton`, także `Checkbutton` pozwala na powiązanie go z obiektem `IntVar`. Jednak w przeciwieństwie do `Radiobutton`, poszczególne widżety `Checkbutton` można wiązać z różnymi obiektami `IntVar`. Po zaznaczeniu pola wyboru powiązany z nim obiekt `IntVar` będzie przechowywał wartość `1`. Jeżeli pole wyboru nie jest zaznaczone, wówczas obiekt ten ma wartość `0`.

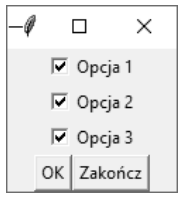
Program przedstawiony na listingu 13.13 pokazuje, jak można utworzyć widżety `Checkbutton` i ich używać. Na rysunku 13.25 pokazałem okno wyświetlone przez ten program. Gdy użytkownik kliknie przycisk *OK*, na ekranie zostanie wyświetlone informacyjne okno dialogowe wskazujące zaznaczone pole wyboru.

Listing 13.13 (checkbutton_demo.py)

```
# Ten program pokazuje grupę widżetów Checkbutton.
import tkinter
import tkinter.messagebox

class MyGUI:
    def __init__(self):
        # Utworzenie okna głównego.
        self.main_window = tkinter.Tk()

        # Utworzenie dwóch kontenerów. Jednego dla widżetów Checkbutton,
        # natomiast drugiego dla zwykłych widżetów Button.
        self.top_frame = tkinter.Frame(self.main_window)
        self.bottom_frame = tkinter.Frame(self.main_window)

        # Utworzenie obiektu IntVar przeznaczonego
        # do użycia wraz z widżetami Checkbuttons.
        self.cb_var1 = tkinter.IntVar()
```

```
        self.cb_var2 = tkinter.IntVar()
        self.cb_var3 = tkinter.IntVar()

        # Przypisanie obiektom IntVar wartości 0.
        self.cb_var1.set(0)
        self.cb_var2.set(0)
        self.cb_var3.set(0)

        # Utworzenie widżetów Checkbutton w kontenerze top_frame.
        self.cb1 = tkinter.Checkbutton(self.top_frame,
                                       text='Opcja 1',
                                       variable=self.cb_var1)
        self.cb2 = tkinter.Checkbutton(self.top_frame,
                                       text='Opcja 2',
                                       variable=self.cb_var2)
        self.cb3 = tkinter.Checkbutton(self.top_frame,
                                       text='Opcja 3',
                                       variable=self.cb_var3)

        # Wywołanie metody pack() widżetów Checkbutton.
        self.cb1.pack()
        self.cb2.pack()
        self.cb3.pack()

        # Utworzenie przycisków OK i Zakończ.
        self.ok_button = tkinter.Button(self.bottom_frame,
                                        text='OK',
                                        command=self.show_choice)
        self.quit_button = tkinter.Button(self.bottom_frame,
                                          text='Zakończ',
                                          command=self.main_window.destroy)

        # Wywołanie metody pack() widżetów Button.
        self.ok_button.pack(side='left')
        self.quit_button.pack(side='left')

        # Wywołanie metody pack() kontenerów.
        self.top_frame.pack()
        self.bottom_frame.pack()

        # Wywołanie metody mainloop() modułu tkinter.
        tkinter.mainloop()

    # Metoda show_choice() jest funkcją
    # wywołania zwrotnego dla przycisku OK.

    def show_choice(self):
        # Utworzenie ciągu tekstowego zawierającego komunikat.
        self.message = 'Wybrane opcje:\n'

        # Ustalenie zaznaczonego pola wyboru i odpowiednie
        # przygotowanie ciągu tekstowego komunikatu.
        if self.cb_var1.get() == 1:
            self.message = self.message + '1\n'
        if self.cb_var2.get() == 1:
            self.message = self.message + '2\n'
        if self.cb_var3.get() == 1:
            self.message = self.message + '3\n'

        # Wyświetlenie komunikatu w informacyjnym oknie dialogowym.
        tkinter.messagebox.showinfo('Wybór', self.message)
```

```
79
80  # Utworzenie egzemplarza klasy MyGUI.
81  my_gui = MyGUI()
```



Rysunek 13.25. Okno wyświetlone przez program z listingu 13.13

Punkt kontrolny

13.15. Chcesz zapewnić użytkownikowi możliwość wyboru tylko jednego elementu z grupy. Jakiego typu komponentu użyjesz do przygotowania tych elementów, przycisku opcji czy pola wyboru?

13.16. Chcesz zapewnić użytkownikowi możliwość wyboru dowolnej liczby elementów z grupy. Jakiego typu komponentu użyjesz do przygotowania tych elementów, przycisku opcji czy pola wyboru?

13.17. Jak można użyć obiektu `IntVar` do ustalenia, który widżet `Radiobutton` został zaznaczony w grupie przycisków opcji?

13.18. Jak można użyć obiektu `IntVar` do ustalenia, który widżet `Checkbutton` został zaznaczony w grupie pól wyboru?

13.9. Tworzenie kształtów z wykorzystaniem widżetu Canvas

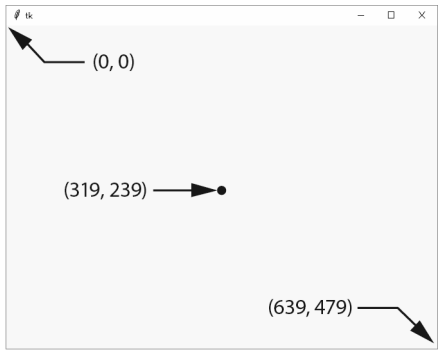
WYJAŚNIENIE. **Widżet `Canvas` oferuje metody przeznaczone do rysowania prostych kształtów, takich jak linie, prostokąty, owale, wielokąty itd.**

Widżet `Canvas` to pusty, prostokątny obszar, na którym można rysować proste kształty dwuwymiarowe. W tym podrozdziale omówię metody oferowane przez ten widżet do rysowania linii, prostokątów, owali, krzywych, wielokątów i tekstu. Zanim przejdę do tematu rysowania wymienionych kształtów, najpierw muszę omówić układ współrzędnych ekranu. Ten **układ współrzędnych ekranu** widżetu `Canvas` jest używany do określenia położenia grafiki.

Układ współrzędnych widżetu Canvas

Obraz wyświetlany na ekranie składa się z małych punktów nazywanych **pikselami**. Układ współrzędnych ekranu jest używany do określenia położenia każdego piksela w oknie aplikacji. Piksel ma współrzędne X i Y. Współrzędna X określa położenie piksela w poziomie, natomiast Y w pionie. Współrzędne są zwykle zapisywane w postaci (X, Y).

W wykorzystywanym przez widżet `Canvas` układzie współrzędnych położenie piksela w lewym górnym rogu to (0, 0). Oznacza to, że obie współrzędne mają wartość 0. Współrzędna X zwiększa wartości od lewej do prawej strony, z kolei współrzędna Y zwiększa je od góry do dołu. Dlatego też w oknie o długości 640 pikseli i wysokości 480 współrzędne piksela znajdującego się w prawym dolnym rogu to (639, 479). W tym samym oknie współrzędne piksela znajdującego się w centrum to (319, 239). Na rysunku 13.26 pokazałem współrzędne różnych pikseli w przykładowym oknie.



Rysunek 13.26. Współrzędne różnych punktów w oknie o wymiarach 640 na 480 pikseli

UWAGA. Układ współrzędnych widżetu `Canvas` różni się od kartezjańskiego układu współrzędnych używanego przez bibliotekę grafiki żółwia. Oto krótkie omówienie różnic.

- W widżecie `Canvas` punkt o współrzędnych (0, 0) wskazuje lewy górny róg ekranu. Natomiast w bibliotece grafiki żółwia ten punkt wskazuje środek ekranu.
- W widżecie `Canvas` współrzędna Y zwiększa się w dół ekranu. Z kolei w bibliotece grafiki żółwia współrzędna Y zmniejsza się w dół ekranu.

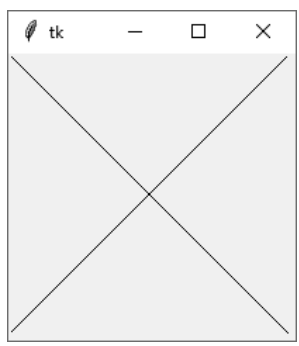
Widżet `Canvas` oferuje wiele metod przeznaczonych do rysowania kształtów na jego powierzchni. W rozdziale omówię następujące metody:

- `create_line()`,
- `create_rectangle()`,
- `create_oval()`,
- `create_arc()`,
- `create_polygon()`,
- `create_text()`.

Zanim przejdę do przedstawienia szczegółów tych metod, spójrz na program zademonstrowany na listingu 13.14. To bardzo prosty program korzystający z widżetu `Canvas` do narysowania linii. Na rysunku 13.27 pokazałem okno wyświetlane przez ten program.

Listing 13.14 (draw_line.py)

```
1  # Ten program pokazuje przykład użycia widżetu Canvas.
2  import tkinter
3
4  class MyGUI:
5      def __init__(self):
6          # Utworzenie okna głównego.
7          self.main_window = tkinter.Tk()
8
9          # Utworzenie widżetu Canvas.
10         self.canvas = tkinter.Canvas(self.main_window, width=200,height=200)
11
12         # Narysowanie dwóch linii.
13         self.canvas.create_line(0, 0, 199, 199)
14         self.canvas.create_line(199, 0, 0, 199)
15
16         # Wywołanie metody pack() widżetu Canvas.
17         self.canvas.pack()
18
19         # Wywołanie metody mainloop() modułu tkinter.
20         tkinter.mainloop()
21
22 # Utworzenie egzemplarza klasy MyGUI.
23 my_gui = MyGUI()
```



Rysunek 13.27. Okno wyświetlone przez program z listingu 13.14

Dokładnie przeanalizuję ten program. W wierszu 10. następuje utworzenie widżetu `Canvas`. Pierwszym argumentem wywołania `Canvas()` jest odwołanie do `self.main_window`, czyli kontenera nadrzędnego, do którego będą dodawane widżety. Argumenty `width=200` i `height=200` określają wielkość widżetu.

W wierszu 13. znajduje się wywołanie metody `create_line()` widżetu `Canvas`. Dwa pierwsze argumenty to współrzędne punktu początkowego linii. Trzeci i czwarty

argument to współrzędne punktu końcowego linii. Dlatego też omówione polecenie powoduje narysowanie w widżecie `Canvas` linii z punktu (0, 0) do punktu (199, 199).

W wierszu 14. również znajduje się wywołanie metody `create_line()` widżetu `Canvas` odpowiedzialne za narysowanie drugiej linii. Tym razem punktem początkowym jest (199, 0), a końcowym (0, 199).

W wierszu 17. znajduje się wywołanie metody `pack()` widżetu `Canvas`, które powoduje wyświetlenie go na ekranie. Natomiast w wierszu 20. wywoływana jest metoda `mainloop()` modułu `tkinter`.

Rysowanie linii — metoda create_line()

Metoda `create_line()` rysuje linię między dwoma lub więcej punktami widżetu `Canvas`. Oto ogólna postać wywołania metody odpowiedzialnej za narysowanie linii między dwoma punktami.

```
widżet_canvas.create_line(x1, y1, x2, y2, opcje...)
```

Argumenty *x1* i *y1* to współrzędne (X, Y) punktu początkowego linii, natomiast *x2* i *y2* to współrzędne (X, Y) punktu końcowego linii. W tej ogólnej postaci *opcje* to opcjonalne argumenty w postaci słów kluczowych, które można przekazać metodzie. Wybrane z nich przedstawię w tabeli 13.2.

Przykłady wywołania metody `create_line()` przedstawiłem w programie na listingu 13.14. Oto polecenie w wierszu 13. rysujące linię od punktu (0, 0) do punktu (199, 199).

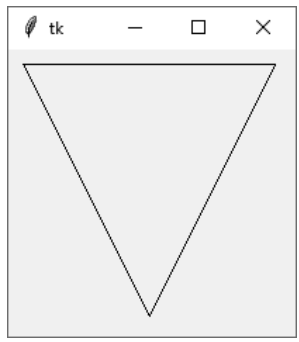
```
self.canvas.create_line(0, 0, 199, 199)
```

Jako argumenty można przekazać wiele zbiorów współrzędnych, a metoda `create_line()` narysuje linie łączące podane punkty. Takie rozwiązanie pokazałem w programie na listingu 13.15. Polecenie w wierszu 13. rysuje linię łączącą punkty (10, 10), (189, 10) (100, 189) i (10, 10). Na rysunku 13.28 znajduje się okno wyświetlane przez ten program.

Listing 13.15 (draw_multi_lines.py)

```
1  # Ten program łączy linią wiele punktów.
2  import tkinter
3
4  class MyGUI:
5      def __init__(self):
6          # Utworzenie okna głównego.
7          self.main_window = tkinter.Tk()
8
9          # Utworzenie widżetu Canvas.
10         self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)
11
12         # Narysowanie linii łączącej wiele punktów.
13         self.canvas.create_line(10, 10, 189, 10, 100, 189, 10, 10)
14
15         # Wywołanie metody pack() widżetu Canvas.
16         self.canvas.pack()
```

```
17
18          # Wywołanie metody mainloop() modułu tkinter.
19          tkinter.mainloop()
20
21  # Utworzenie egzemplarza klasy MyGUI.
22  my_gui = MyGUI()
```



Rysunek 13.28. Okno wyświetlone przez program z listingu 13.15

Ewentualnie metodzie można przekazać argument w postaci listy lub krotki zawierającej współrzędne. Dlatego też w programie przedstawionym na listingu 13.15 polecenie w wierszu 13. można zastąpić następującym fragmentem kodu i otrzymać ten sam wynik:

```
points = [10, 10, 189, 10, 100, 189, 10, 10]
self.canvas.create_line(points)
```

Istnieje wiele opcjonalnych argumentów w postaci słów kluczowych, które można przekazać metodzie `create_line()`. W tabeli 13.2 wymieniłem najczęściej używane.

Rysowanie prostokąta — metoda create_rectangle()

Metoda `create_rectangle()` powoduje narysowanie prostokąta w widżecie `Canvas`. Oto ogólna postać wywołania tej metody.

```
widżet_canvas.create_rectangle(x1, y1, x2, y2, opcje...)
```

Argumenty *x1* i *y1* to współrzędne (X, Y) lewego górnego wierzchołka, natomiast *x2* i *y2* to współrzędne (X, Y) prawego dolnego wierzchołka prostokąta. W tej ogólnej postaci *opcje* to opcjonalne argumenty w postaci słów kluczowych, które można przekazać metodzie. Wybrane z nich przedstawię w tabeli 13.3.

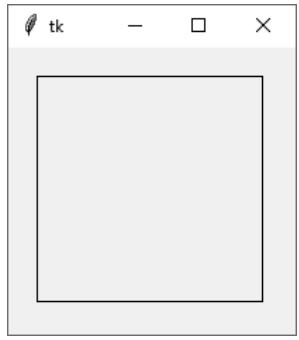
Program na listingu 13.16 pokazuje w wierszu 13. przykład wywołania metody `create_rectangle()`. Lewy górny wierzchołek znajduje się w punkcie o współrzędnych (20, 20), a dolny prawy w punkcie (180, 180). Na rysunku 13.29 pokazałem okno wyświetlane przez ten program.

Tabela 13.2. Wybrane argumenty opcjonalne metody create_line()

Argument	Opis
`arrow=`*`wartość`*	Domyślnie rysowane linie nie mają strzałek. Przy użyciu tego argumentu można nakazać narysowanie strzałki na jednym lub obu końcach linii. Argument `arrow.tk.FIRST` powoduje narysowanie strzałki na początku linii, natomiast `arrow.tk.LAST` rysuje strzałkę na końcu linii. Jeżeli chcesz otrzymać strzałkę na obu końcach linii, musisz użyć argumentu `arrow.tk.BOTH`.
`dash=`*`wartość`*	Argument powoduje narysowanie przerywanej linii. Wartością tego argumentu jest krotka zawierająca liczby całkowite określające wzorzec. Pierwszy element krotki podaje liczbę pikseli do narysowania, drugi liczbę pikseli do pominięcia itd. Przykładowo argument `dash=(5, 2)` spowoduje narysowanie pięciu pikseli, pominięcie dwóch i powtarzanie tego wzorca aż do końca linii.
`fill=`*`wartość`*	Określa kolor linii. Wartością argumentu jest nazwa koloru przedstawiona w postaci ciągu tekstowego. Istnieje wiele predefiniowanych nazw kolorów, których można użyć. Pełną listę nazw kolorów przedstawiłem w dodatku D. Najczęściej używane nazwy kolorów to `'red'`, `'green'`, `'blue'`, `'yellow'` i `'cyan'`. (Jeśli pominiesz argument `fill`, domyślnym kolorem wypełnienia będzie czarny).
`smooth=`*`wartość`*	Domyślnie ten argument ma wartość `False`, co powoduje, że metoda `create_line()` rysuje linię prostą łączącą podane punkty. Jeżeli użyjesz argumentu `smooth=True`, rysowane linie będą zaokrąglone.
`width=`*`wartość`*	Argument określa szerokość linii w pikselach. Przykładowo argument `width=5` oznacza, że linia będzie miała 5 pikseli szerokości. Domyślną szerokością linii jest 1 piksel.

Listing 13.16 (draw_square.py)

```
# Ten program rysuje prostokąt w widżecie Canvas.
import tkinter

class MyGUI:
    def __init__(self):
        # Utworzenie okna głównego.
        self.main_window = tkinter.Tk()

        # Utworzenie widżetu Canvas.
        self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)

        # Narysowanie prostokąta.
        self.canvas.create_rectangle(20, 20, 180, 180)

        # Wywołanie metody pack() widżetu Canvas.
        self.canvas.pack()

        # Wywołanie metody mainloop() modułu tkinter.
        tkinter.mainloop()

# Utworzenie egzemplarza klasy MyGUI.
my_gui = MyGUI()
```



Rysunek 13.29. Okno wyświetlone przez program z listingu 13.16

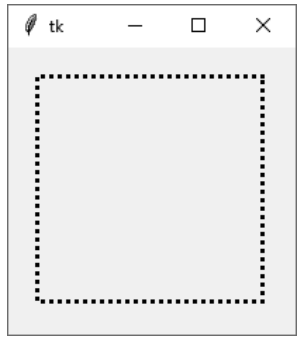
Istnieje wiele opcjonalnych argumentów w postaci słów kluczowych, które można przekazać metodzie `create_rectangle()`. W tabeli 13.3 wymieniłem najczęściej używane.

Tabela 13.3. Wybrane argumenty opcjonalne metody create_rectangle()

Argument	Opis
`dash=`*wartość*	Argument powoduje użycie przerywanej linii podczas rysowania prostokąta. Wartością tego argumentu jest krotka zawierająca liczby całkowite określające wzorzec. Pierwszy element krotki podaje liczbę pikseli do narysowania, drugi liczbę pikseli do pominięcia itd. Przykładowo argument `dash=(5, 2)` spowoduje narysowanie pięciu pikseli, pominięcie dwóch i powtarzanie tego wzorca aż do końca linii.
`fill=`*wartość*	Określa kolor wypełnienia prostokąta. Wartością argumentu jest nazwa koloru przedstawiona w postaci ciągu tekstowego. Istnieje wiele predefiniowanych nazw kolorów, których można używać. Pełną listę nazw kolorów przedstawiłem w dodatku D. Najczęściej używane nazwy kolorów to `'red'`, `'green'`, `'blue'`, `'yellow'` i `'cyan'`. (Jeśli pominiesz argument `fill`, domyślnym kolorem wypełnienia będzie czarny).
`outline=`*wartość*	Argument określa kolor obramowania prostokąta. Wartością argumentu jest nazwa koloru przedstawiona w postaci ciągu tekstowego. Istnieje wiele predefiniowanych nazw kolorów, których można używać. Pełną listę nazw kolorów przedstawiłem w dodatku D. Najczęściej używane nazwy kolorów to `'red'`, `'green'`, `'blue'`, `'yellow'` i `'cyan'`. (Jeśli pominiesz argument `outline`, kolorem domyślnym będzie czarny).
`width=`*wartość*	Argument określa szerokość linii w pikselach. Przykładowo argument `width=5` oznacza, że linia będzie miała 5 pikseli szerokości. Domyślną szerokością linii jest 1 piksel.

Jeśli np. zmodyfikujesz program przedstawiony na listingu 13.16, prostokąt zostanie narysowany za pomocą przerywanej linii o odstępach wynoszących 3 piksele. Na rysunku 13.30 pokazałem okno wygenerowane przez zmodyfikowaną wersję programu, którego wiersz 13. ma następującą postać:

```
self.canvas.create_rectangle(20, 20, 180, 180, dash=(5, 2), width=3)
```



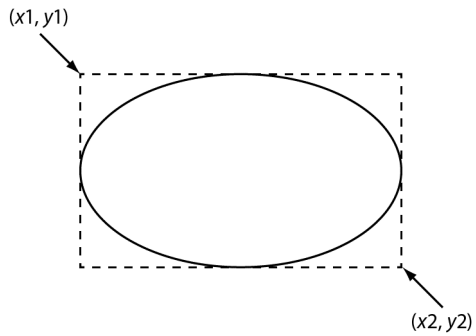
Rysunek 13.30. Okno wyświetlone przez zmodyfikowaną wersję programu z listingu 13.16

Rysowanie owalu — metoda create_oval()

Metoda `create_oval()` powoduje narysowanie owalu w widżecie `Canvas`. Oto ogólna postać wywołania tej metody.

```
widżet_canvas.create_oval(x1, y1, x2, y2, opcje...)
```

Metoda rysuje owal doskonale wpasowujący się w prostokąt zdefiniowany przez współrzędne przekazane jako argumenty tej metody. Argumenty *x1* i *y1* to współrzędne (X, Y) lewego górnego wierzchołka, natomiast *x2* i *y2* to współrzędne (X, Y) prawego dolnego wierzchołka prostokąta. Pokazałem to na rysunku 13.31. W tej ogólnej postaci *opcje* to opcjonalne argumenty w postaci słów kluczowych, które można przekazać metodzie. Wybrane z nich przedstawię w tabeli 13.4.

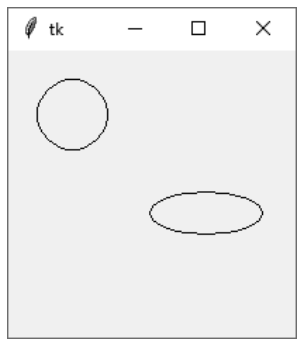


Rysunek 13.31. Prostokąt, w który jest wpasowany owal

Program na listingu 13.17 pokazuje w wierszach 13. i 14. przykłady użycia metody `create_oval()`. Pierwszy owal narysowany w wierszu 13. mieści się w prostokącie zdefiniowanym przez lewy górny wierzchołek w punkcie o współrzędnych (20, 20) i dolny prawy wierzchołek w punkcie (70, 70). Z kolei drugi owal narysowany w wierszu 14. mieści się w prostokącie zdefiniowanym przez lewy górny wierzchołek w punkcie o współrzędnych (100, 100) i dolny prawy wierzchołek w punkcie (180, 130). Na rysunku 13.32 pokazałem okno wyświetlane przez ten program.

Listing 13.17 (draw_ovals.py)

```
1  # Ten program pokazuje narysowanie dwóch owali w widżecie Canvas.
2  import tkinter
3
4  class MyGUI:
5      def __init__(self):
6          # Utworzenie okna głównego.
7          self.main_window = tkinter.Tk()
8
9          # Utworzenie widżetu Canvas.
10         self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)
11
12         # Narysowanie dwóch owali.
13         self.canvas.create_oval(20, 20, 70, 70)
14         self.canvas.create_oval(100, 100, 180, 130)
15
16         # Wywołanie metody pack() widżetu Canvas.
17         self.canvas.pack()
18
19         # Wywołanie metody mainloop() modułu tkinter.
20         tkinter.mainloop()
21
22 # Utworzenie egzemplarza klasy MyGUI.
23 my_gui = MyGUI()
```



Rysunek 13.32. Okno wyświetlone przez program z listingu 13.17

Istnieje wiele opcjonalnych argumentów w postaci słów kluczowych, które można przekazać metodzie `create_oval()`. W tabeli 13.4 wymieniłem najczęściej używane.

WSKAZÓWKA. Jeżeli chcesz narysować okrąg, wywołaj metodę `create_oval()` wraz z argumentami definiującymi taką samą długość wszystkich boków prostokąta.

Listing 13.4. Wybrane argumenty opcjonalne metody create_oval()

Argument	Opis
`dash=`*`wartość`*	Argument powoduje użycie przerywanej linii podczas rysowania owalu. Wartością tego argumentu jest krotka zawierająca liczby całkowite określające wzorzec. Pierwszy element krotki podaje liczbę pikseli do narysowania, drugi liczbę pikseli do pominięcia itd. Przykładowo argument `dash=(5, 2)` spowoduje narysowanie pięciu pikseli, pominięcie dwóch i powtarzanie tego wzorca aż do końca linii.
`fill=`*`wartość`*	Określa kolor wypełnienia owalu. Wartością argumentu jest nazwa koloru przedstawiona w postaci ciągu tekstowego. Istnieje wiele predefiniowanych nazw kolorów, których można używać. Pełną listę nazw kolorów przedstawiłem w dodatku D. Najczęściej używane nazwy kolorów to `'red'`, `'green'`, `'blue'`, `'yellow'` i `'cyan'`. (Jeśli pominiesz argument `fill`, domyślnym kolorem wypełnienia będzie czarny).
`outline=`*`wartość`*	Argument określa kolor obramowania owalu. Wartością argumentu jest nazwa koloru przedstawiona w postaci ciągu tekstowego. Istnieje wiele predefiniowanych nazw kolorów, których można używać. Pełną listę nazw kolorów przedstawiłem w dodatku D. Najczęściej używane nazwy kolorów to `'red'`, `'green'`, `'blue'`, `'yellow'` i `'cyan'`. (Jeśli pominiesz argument `outline`, kolorem domyślnym będzie czarny).
`width=`*`wartość`*	Argument określa szerokość linii w pikselach. Przykładowo argument `width=5` oznacza, że linia będzie miała 5 pikseli szerokości. Domyślną szerokością linii jest 1 piksel.

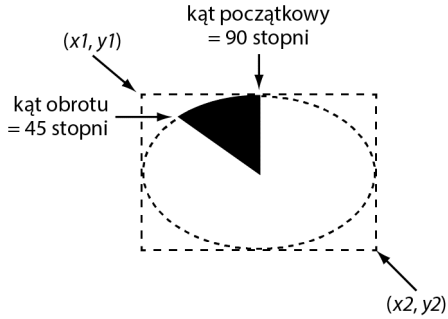
Rysowanie wycinka koła — metoda create_arc()

Metoda `create_arc()` powoduje narysowanie wycinka koła w widżecie `Canvas`. Oto ogólna postać wywołania tej metody.

```
widżet_canvas.create_arc(x1, y1, x2, y2, start=kąt, extent=długość, opcje...)
```

Metoda powoduje narysowanie wycinka koła będącego fragmentem owalu doskonale wpasowującego się w prostokąt zdefiniowany przez współrzędne przekazane jako argumenty tej metody. Argumenty *x1* i *y1* to współrzędne (X, Y) lewego górnego wierzchołka, natomiast *x2* i *y2* to współrzędne (X, Y) prawego dolnego wierzchołka prostokąta. Argument `start=`*`kąt`* określa kąt początkowy, a `extent=`*`długość`* wskazuje kąt obrotu w kierunku przeciwnym do ruchu wskazówek zegara. Przykładowo argument `start=90` określa kąt początkowy wynoszący 90 stopni. Z kolei argument `extent=45` wskazuje kąt obrotu wynoszący 45 stopni w kierunku przeciwnym do ruchu wskazówek zegara. Pokazałem to na rysunku 13.33.

W tej ogólnej postaci *opcje* to opcjonalne argumenty w postaci słów kluczowych, które można przekazać metodzie. Wybrane z nich przedstawię w tabeli 13.5, nieco dalej w rozdziale.



Rysunek 13.33. Argument metody create_arc()

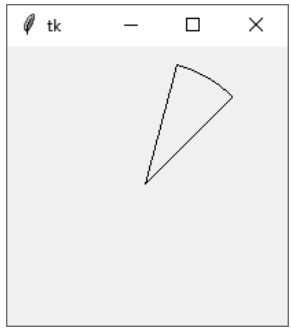
Program na listingu 13.18 pokazuje przykład użycia metody `create_arc()`. Wycinek koła narysowany w wierszu 13. mieści się w prostokącie zdefiniowanym przez lewy górny wierzchołek w punkcie o współrzędnych (10, 10) i dolny prawy wierzchołek w punkcie (190, 190). Narysowany tutaj wycinek koła ma kąt początkowy wynoszący 45 stopni i kąt obrotu 30 stopni. Na rysunku 13.34 pokazałem okno wyświetlane przez ten program.

Listing 13.18 (draw_arc.py)

```
1  # Ten program pokazuje przykład narysowania wycinka koła w widżecie Canvas.
2  import tkinter
3
4  class MyGUI:
5      def __init__(self):
6          # Utworzenie okna głównego.
7          self.main_window = tkinter.Tk()
8
9          # Utworzenie widżetu Canvas.
10         self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)
11
12         # Narysowanie wycinka koła.
13         self.canvas.create_arc(10, 10, 190, 190, start=45, extent=30)
14
15         # Wywołanie metody pack() widżetu Canvas.
16         self.canvas.pack()
17
18         # Wywołanie metody mainloop() modułu tkinter.
19         tkinter.mainloop()
20
21 # Utworzenie egzemplarza klasy MyGUI.
22 my_gui = MyGUI()
```

Istnieje wiele opcjonalnych argumentów w postaci słów kluczowych, które można przekazać metodzie `create_arc()`. W tabeli 13.5 wymieniłem najczęściej używane.

Argument `style=`*`wartość`* pozwala na użycie różnych stylów podczas rysowania wycinka koła. Podsumowanie dostępnych stylów przedstawiłem w tabeli 13.6. (Zwróć uwagę na to, że stylem domyślnym jest `tkinter.PIESLICE`). Przykłady poszczególnych stylów pokazałem na rysunku 13.35.



Rysunek 13.34. Okno wyświetlone przez program z listingu 13.18

Tabela 13.5. Wybrane argumenty opcjonalne metody create_arc()

Argument	Opis
`dash=`*wartość*	Argument powoduje użycie przerywanej linii podczas rysowania wycinka koła. Wartością tego argumentu jest krotka zawierająca liczby całkowite określające wzorzec. Pierwszy element krotki podaje liczbę pikseli do narysowania, drugi liczbę pikseli do pominięcia itd. Przykładowo argument `dash=(5, 2)` spowoduje narysowanie pięciu pikseli, pominięcie dwóch i powtarzanie tego wzorca aż do końca linii.
`fill=`*wartość*	Określa kolor wypełnienia narysowanego wycinka koła. Wartością argumentu jest nazwa koloru przedstawiona w postaci ciągu tekstowego. Istnieje wiele predefiniowanych nazw kolorów, których można używać. Pełną listę nazw kolorów przedstawiłem w dodatku D. Najczęściej używane nazwy kolorów to `'red'`, `'green'`, `'blue'`, `'yellow'` i `'cyan'`. (Jeśli pominiesz argument `fill`, kształt nie będzie wypełniony żadnym kolorem).
`outline=`*wartość*	Argument określa kolor obramowania wycinka koła. Wartością argumentu jest nazwa koloru przedstawiona w postaci ciągu tekstowego. Istnieje wiele predefiniowanych nazw kolorów, których można używać. Pełną listę nazw kolorów przedstawiłem w dodatku D. Najczęściej używane nazwy kolorów to `'red'`, `'green'`, `'blue'`, `'yellow'` i `'cyan'`. (Jeśli pominiesz argument `outline`, kolorem domyślnym będzie czarny).
`style=`*wartość*	Argument określa styl narysowanego wycinka koła. Wartością argumentu może być `tkinter.PIESLICE`, `tkinter.ARC` lub `tkinter.CHORD`. Więcej informacji na ten temat znajdziesz w tabeli 13.6.
`width=`*wartość*	Argument określa szerokość linii w pikselach. Przykładowo argument `width=5` oznacza, że linia będzie miała 5 pikseli szerokości. Domyślną szerokością linii jest 1 piksel.

Rysunek 13.35. Style dostępne podczas rysowania wycinka koła

Tabela 13.6. Style stosowane podczas rysowania wycinka koła

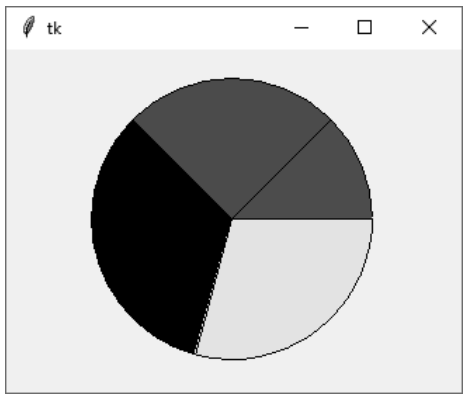
Argument style	Opis
`style=tkinter.PIESLICE`	To jest domyślny styl wycinka koła. Linie proste zostaną narysowane od środka kształtu do każdego punktu końcowego. W efekcie powstanie kształt przypominający porcję ciasta (ang. *pie slice*).
`style=tkinter.ARC`	Brak linii łączących punkty końcowe. W efekcie powstanie sam łuk.
`style=tkinter.CHORD`	Zwykła linia połączy punkty końcowe.

Program na listingu 13.19 pokazuje przykład użycia stylu `tkinter.PIESLICE`. Na rysunku 13.36 pokazałem okno wyświetlane przez ten program.

Listing 13.19 (draw_piechart.py)

```
# Ten program pokazuje przykłady użycia metody create_arc() w widżecie Canvas.
import tkinter

class MyGUI:
    def __init__(self):
        self.__CANVAS_WIDTH = 320  # Długość widżetu Canvas.
        self.__CANVAS_HEIGHT = 240 # Wysokość widżetu Canvas.
        self.__X1 = 60             # Współrzędna X lewego górnego wierzchołka prostokąta.
        self.__Y1 = 20             # Współrzędna Y lewego górnego wierzchołka prostokąta.
        self.__X2 = 260            # Współrzędna X prawego dolnego wierzchołka prostokąta.
        self.__Y2 = 220            # Współrzędna Y prawego dolnego wierzchołka prostokąta.
        self.__PIE1_START = 0      # Kąt początkowy dla kształtu nr 1.
        self.__PIE1_WIDTH = 45     # Kąt obrotu dla kształtu nr 1.
        self.__PIE2_START = 45     # Kąt początkowy dla kształtu nr 2.
        self.__PIE2_WIDTH = 90     # Kąt obrotu dla kształtu nr 2.
        self.__PIE3_START = 135    # Kąt początkowy dla kształtu nr 3.
        self.__PIE3_WIDTH = 120    # Kąt obrotu dla kształtu nr 3.
        self.__PIE4_START = 255    # Kąt początkowy dla kształtu nr 4.
        self.__PIE4_WIDTH = 105    # Kąt obrotu dla kształtu nr 4.

        # Utworzenie okna głównego.
        self.main_window = tkinter.Tk()

        # Utworzenie widżetu Canvas.
        self.canvas = tkinter.Canvas(self.main_window,
                                     width=self.__CANVAS_WIDTH,
                                     height=self.__CANVAS_HEIGHT)

        # Narysowanie kształtu nr 1.
        self.canvas.create_arc(self.__X1, self.__Y1, self.__X2, self.__Y2,
                               start=self.__PIE1_START,
                               extent=self.__PIE1_WIDTH,
                               fill='red')

        # Narysowanie kształtu nr 2.
        self.canvas.create_arc(self.__X1, self.__Y1, self.__X2, self.__Y2,
                               start=self.__PIE2_START,
                               extent=self.__PIE2_WIDTH,
                               fill='green')

        # Narysowanie kształtu nr 3.
        self.canvas.create_arc(self.__X1, self.__Y1, self.__X2, self.__Y2,
```

```
43                              start=self.__PIE3_START,
44                              extent=self.__PIE3_WIDTH,
45                              fill='black')
46
47          # Narysowanie kształtu nr 4.
48          self.canvas.create_arc(self.__X1, self.__Y1, self.__X2, self.__Y2,
49                              start=self.__PIE4_START,
50                              extent=self.__PIE4_WIDTH,
51                              fill='yellow')
52
53          # Wywołanie metody pack() widżetu Canvas.
54          self.canvas.pack()
55
56          # Wywołanie metody mainloop() modułu tkinter.
57          tkinter.mainloop()
58
59  # Utworzenie egzemplarza klasy MyGUI.
60  my_gui = MyGUI()
```



Rysunek 13.36. Okno wyświetlone przez program z listingu 13.19

Przeanalizuję teraz dokładnie metodę `__init__()` klasy `MyGUI` w programie przedstawionym na listingu 13.19.

- W wierszach 6. i 7. zostały zdefiniowane atrybuty określające długość i wysokość widżetu `Canvas`.
- W wierszach od 8. do 11. zostały zdefiniowane atrybuty dla współrzędnych lewego górnego i prawego dolnego wierzchołka prostokąta, w który jest wpasowany owal, którego z kolei fragmentem jest tworzony kształt.
- W wierszach od 12. do 19. zostały zdefiniowane atrybuty dla kątów początkowego i obrotu poszczególnych kształtów.
- W wierszu 22. zostało utworzone okno główne, natomiast w wierszach od 25. do 27. widżet `Canvas`.
- W wierszach od 30. do 33. powstaje pierwszy kształt wypełniony kolorem czerwonym.
- W wierszach od 36. do 39. powstaje drugi kształt wypełniony kolorem zielonym.

- W wierszach od 42. do 45. powstaje trzeci kształt wypełniony kolorem czarnym.
- W wierszach od 48. do 51. powstaje czwarty kształt wypełniony kolorem żółtym.
- Wiersz 54. to wywołanie metody `pack()` widżetu `Canvas`, zaś wiersz 57. zawiera wywołanie funkcji `mainloop()` modułu `tkinter`.

Rysowanie wielokąta — metoda create_polygon()

Metoda `create_polygon()` powoduje narysowanie zamkniętego wielokąta w widżecie `Canvas`. Wielokąt ten składa się z wielu połączonych segmentów linii. Punkt przecięcia dwóch linii jest nazywany **wierzchołkiem**. Oto ogólna postać wywołania tej metody.

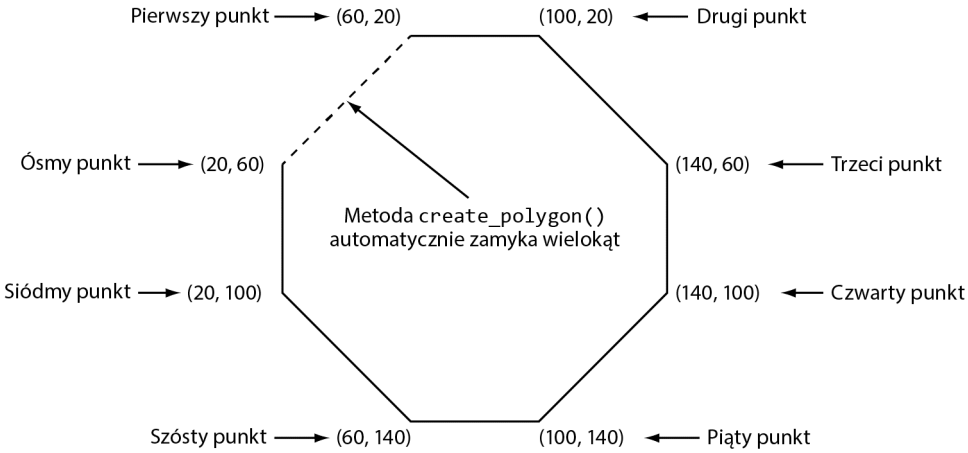
```
widżet_canvas.create_polygon(x1, y1, x2, y2, ..., opcje...)
```

Argumenty *x1* i *y1* to współrzędne (X, Y) pierwszego wierzchołka, natomiast *x2* i *y2* to współrzędne (X, Y) drugiego wierzchołka itd. Metoda automatycznie zamknie wielokąt przez narysowanie linii łączącej pierwszy i ostatni wierzchołek. W tej ogólnej postaci *opcje* to opcjonalne argumenty w postaci słów kluczowych, które można przekazać metodzie. Wybrane z nich przedstawię w tabeli 13.7 nieco dalej w rozdziale.

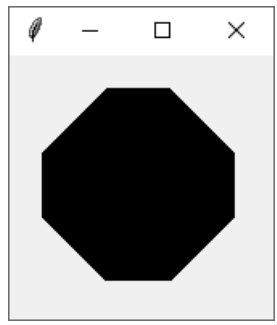
Program na listingu 13.20 pokazuje przykład użycia metody `create_polygon()`. Polecenie w wierszach 13. i 14. rysuje wielokąt o ośmiu wierzchołkach. Pierwszy wierzchołek znajduje się w punkcie o współrzędnych (60, 20), drugi w punkcie (100, 20) itd., tak jak pokazałem na rysunku 13.37. Natomiast na rysunku 13.38 można zobaczyć okno wyświetlane przez ten program.

Listing 13.20 (draw_polygon.py)

```
1  # Ten program pokazuje przykład narysowania wielokąta w widżecie Canvas.
2  import tkinter
3
4  class MyGUI:
5      def __init__(self):
6          # Utworzenie okna głównego.
7          self.main_window = tkinter.Tk()
8
9          # Utworzenie widżetu Canvas.
10         self.canvas = tkinter.Canvas(self.main_window, width=160, height=160)
11
12         # Narysowanie wielokąta.
13         self.canvas.create_polygon(60, 20, 100, 20, 140, 60, 140, 100,
14                                    100, 140, 60, 140, 20, 100, 20, 60)
15
16         # Wywołanie metody pack() widżetu Canvas.
17         self.canvas.pack()
18
19        # Wywołanie metody mainloop() modułu tkinter.
20        tkinter.mainloop()
21
22 # Utworzenie egzemplarza klasy MyGUI.
23 my_gui = MyGUI()
```



Rysunek 13.37. Punkty poszczególnych wierzchołków wielokąta



Rysunek 13.38. Okno wyświetlone przez program z listingu 13.20

Istnieje wiele opcjonalnych argumentów w postaci słów kluczowych, które można przekazać metodzie `create_polygon()`. W tabeli 13.7 wymieniłem najczęściej używane.

Wyświetlenie tekstu — metoda create_text()

Metoda `create_text()` powoduje wyświetlenie tekstu w widżecie `Canvas`. Oto ogólna postać wywołania tej metody.

```
widżet_canvas.create_text(x, y, text=tekst, opcje...)
```

Argumenty *x* i *y* są współrzędnymi (X, Y) punktu wstawienia tekstu, natomiast *text* definiuje tekst przeznaczony do wyświetlenia. Domyślnie tekst zostanie wyśrodkowany poziomo i pionowo wokół punktu jego wstawienia. W tej ogólnej postaci *opcje* to opcjonalne argumenty w postaci słów kluczowych, które można przekazać metodzie. Wybrane z nich przedstawię w tabeli 13.8 nieco dalej w rozdziale.

Program na listingu 13.21 pokazuje przykład użycia metody `create_text()`. Polecenie w wierszu 13. wyświetla tekst `Witaj, świecie!` w środku okna, w punkcie o współrzędnych (100, 100). Na rysunku 13.39 pokazałem okno wyświetlane przez ten program.

Tabela 13.7. Wybrane argumenty opcjonalne metody create_polygon()

Argument	Opis
`dash=`*wartość*	Argument powoduje użycie przerywanej linii podczas rysowania wielokąta. Wartością tego argumentu jest krotka zawierająca liczby całkowite określające wzorzec. Pierwszy element krotki podaje liczbę pikseli do narysowania, drugi liczbę pikseli do pominięcia itd. Przykładowo argument `dash=(5, 2)` spowoduje narysowanie pięciu pikseli, pominięcie dwóch i powtarzanie tego wzorca aż do końca linii.
`fill=`*wartość*	Określa kolor wypełnienia wielokąta. Wartością argumentu jest nazwa koloru przedstawiona w postaci ciągu tekstowego. Istnieje wiele predefiniowanych nazw kolorów, których można używać. Pełną listę nazw kolorów przedstawiłem w dodatku D. Najczęściej używane nazwy kolorów to `'red'`, `'green'`, `'blue'`, `'yellow'` i `'cyan'`. (Jeśli pominiesz argument `fill`, domyślnym kolorem wypełnienia będzie czarny).
`outline=`*wartość*	Argument określa kolor obramowania wielokąta. Wartością argumentu jest nazwa koloru przedstawiona w postaci ciągu tekstowego. Istnieje wiele predefiniowanych nazw kolorów, których można używać. Pełną listę nazw kolorów przedstawiłem w dodatku D. Najczęściej używane nazwy kolorów to `'red'`, `'green'`, `'blue'`, `'yellow'` i `'cyan'`. (Jeśli pominiesz argument `outline`, kolorem domyślnym będzie czarny).
`smooth=`*wartość*	Domyślnie argument ma wartość `False`, co powoduje, że metoda `create_line()` rysuje linię prostą łączącą podane punkty. Jeżeli użyjesz argumentu `smooth=True`, rysowane linie będą zaokrąglone.
`width=`*wartość*	Argument określa szerokość linii w pikselach. Przykładowo argument `width=5` oznacza, że linia będzie miała 5 pikseli szerokości. Domyślną szerokością linii jest 1 piksel.

Listing 13.21 (draw_text.py)

```
# Ten program pokazuje przykład wyświetlenia tekstu w widżecie Canvas.
import tkinter

class MyGUI:
    def __init__(self):
        # Utworzenie okna głównego.
        self.main_window = tkinter.Tk()

        # Utworzenie widżetu Canvas.
        self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)

        # Wyświetlenie tekstu na środku okna.
        self.canvas.create_text(100, 100, text='Witaj, świecie!')

        # Wywołanie metody pack() widżetu Canvas.
        self.canvas.pack()

        # Wywołanie metody mainloop() modułu tkinter.
        tkinter.mainloop()
```

```
20
21  # Utworzenie egzemplarza klasy MyGUI.
22  my_gui = MyGUI()
```

Rysunek 13.39. Okno wyświetlone przez program z listingu 13.21

Istnieje wiele opcjonalnych argumentów w postaci słów kluczowych, które można przekazać metodzie `create_text()`. W tabeli 13.8 wymieniłem najczęściej używane.

Tabela 13.8. Wybrane argumenty opcjonalne metody create_text()

Argument	Opis
`anchor=`*`wartość`*	Argument określa sposób umieszczenia tekstu względem jego punktu wstawienia. Domyślnie argument `anchor` ma wartość `tkinter.CENTER` powodującą wyśrodkowanie tekstu zarówno w poziomie, jak i w pionie względem punktu wstawienia. Jako wartości omawianego argumentu można użyć dowolnej z wymienionych w tabeli 13.9 nieco dalej w rozdziale.
`fill=`*`wartość`*	Określa kolor tekstu. Wartością argumentu jest nazwa koloru przedstawiona w postaci ciągu tekstowego. Istnieje wiele predefiniowanych nazw kolorów, których można używać. Pełną listę nazw kolorów przedstawiłem w dodatku D. Najczęściej używane nazwy kolorów to `'red'`, `'green'`, `'blue'`, `'yellow'` i `'cyan'`. (Jeśli pominiesz argument `fill`, domyślnym kolorem wypełnienia będzie czarny).
`font=`*`wartość`*	Argument pozwala na zmianę czcionki domyślnej. Wystarczy utworzyć obiekt `tkinter.font.Font` i przekazać go jako wartość omawianego argumentu. (Więcej informacji na temat czcionek znajdziesz dalej w tym rozdziale).
`justify=`*`wartość`*	Jeżeli ma zostać wyświetlonych wiele wierszy tekstu, ten argument określa sposób ich wyrównania. Dostępne wartości to `tk.LEFT`, `tk.CENTER` i `tk.RIGHT`. Wartością domyślną jest `tk.LEFT`.

Mamy dziewięć różnych sposobów na umieszczenie tekstu względem jego punktu wstawienia. Zmiana położenia odbywa się za pomocą argumentu `anchor=`*`położenie`*.

Możliwe do użycia wartości tego argumentu wymieniłem w tabeli 13.9. Zwróć uwagę, że wartością domyślną jest `tkinter.CENTER`.

Tabela 13.9. Wartości argumentu anchor

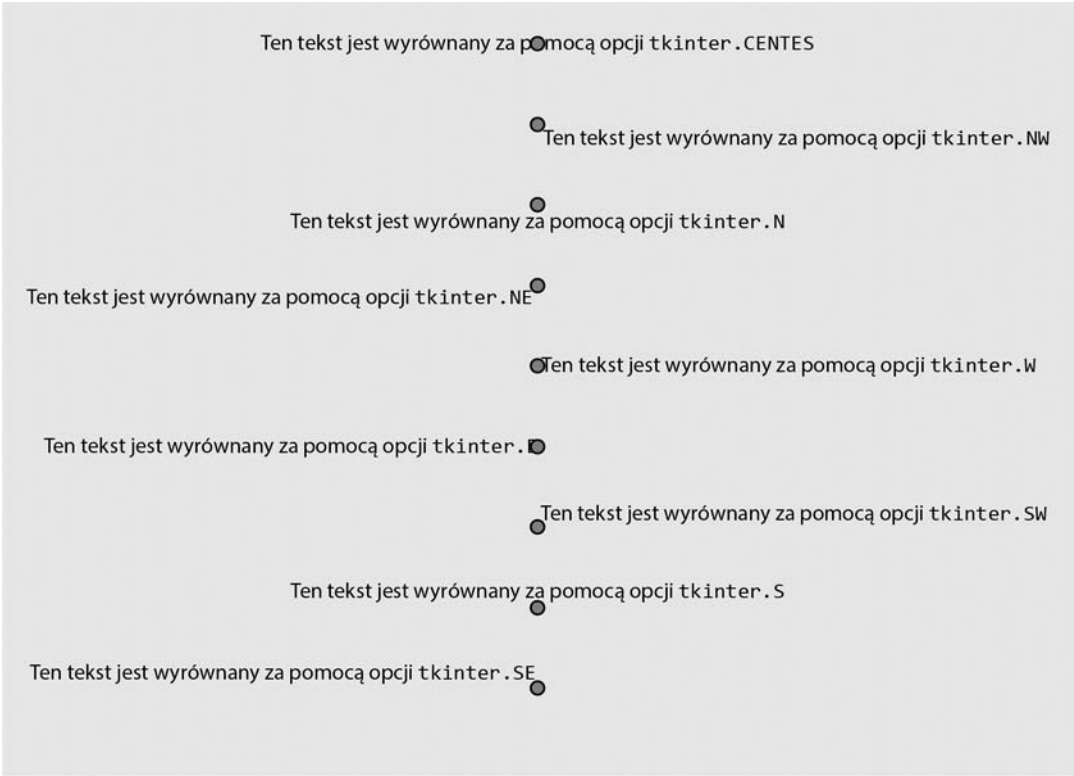
Argument anchor	Opis
`anchor=tkinter.CENTER`	Tekst zostanie wyśrodkowany zarówno poziomo, jak i pionowo względem punktu jego wstawienia. Jest to położenie domyślne.
`anchor=tkinter.NW`	Tekst zostanie umieszczony w taki sposób, aby punkt wstawienia znajdował się w jego lewym górnym wierzchołku — północny zachód (ang. *northwest*, NW).
`anchor=tkinter.N`	Tekst zostanie umieszczony w taki sposób, aby punkt wstawienia był wyrównany względem górnej krawędzi tekstu — północ (ang. *north*, N).
`anchor=tkinter.NE`	Tekst zostanie umieszczony w taki sposób, aby punkt wstawienia znajdował się w jego prawym górnym wierzchołku — północny wschód (ang. *northeast*, NE).
`anchor=tkinter.W`	Tekst zostanie umieszczony w taki sposób, aby punkt wstawienia był wyrównany względem lewej krawędzi tekstu, pośrodku — zachód (ang. *west*, W).
`anchor=tkinter.E`	Tekst zostanie umieszczony w taki sposób, aby punkt wstawienia był wyrównany względem prawej krawędzi tekstu, pośrodku — wschód (ang. *east*, E).
`anchor=tkinter.SW`	Tekst zostanie umieszczony w taki sposób, aby punkt wstawienia znajdował się w jego lewym dolnym wierzchołku — południowy zachód (ang. *southwest*, SW).
`anchor=tkinter.S`	Tekst zostanie umieszczony w taki sposób, aby punkt wstawienia był wyrównany względem dolnej krawędzi tekstu — południe (ang. *south*, S).
`anchor=tkinter.SE`	Tekst zostanie umieszczony w taki sposób, aby punkt wstawienia znajdował się w jego prawym dolnym wierzchołku — południowy wschód (ang. *southeast*, SE).

Na rysunku 13.40 pokazałem efekt działania różnych wartości argumentu `anchor`. Każdy wiersz tekstu oznaczono kropką przedstawiającą położenie punktu wstawienia.

Wybór czcionki

Istnieje możliwość wyboru czcionki używanej wraz z metodą `create_text()`. Wymaga to utworzenia obiektu `Font` i przekazania go jako argumentu `font`. Klasa `Font` została zdefiniowana w module `tkinter.font`, więc w programie trzeba umieścić następujące polecenie:

```
import tkinter.font
```



Rysunek 13.40. Wyniki użycia różnych wartości argumentu anchor

Oto przykład polecenia tworzącego obiekt `Font` określającego czcionkę z rodziny *Helvetica* o wielkości 12 punktów:

```
myfont = tkinter.font.Font(family='Helvetica', size='12')
```

Podczas tworzenia obiektu `Font` można przekazać wartości w postaci słów kluczowych dla dowolnego z wymienionych w tabeli 13.10 argumentów.

Nazwy dostępnych rodzin czcionek zależą od używanego systemu operacyjnego. Aby wyświetlić listę zainstalowanych rodzin czcionek, w trybie interaktywnym Pythona wydaj następujące polecenia:

```
>>> import tkinter
>>> import tkinter.font
>>> tkinter.Tk()
<tkinter.Tk object .>
>>> tkinter.font.families()
```

Na listingu 13.22 przedstawiłem program wyświetlający tekst za pomocą pogrubionej czcionki Helvetica o wielkości 18 punktów. Na rysunku 13.41 pokazałem okno wyświetlane przez ten program.

Tabela 13.10. Argumenty w postaci słów kluczowych używane wraz z obiektem klasy Font

Argument	Opis
`family=`*`wartość`*	To jest ciąg tekstowy określający nazwę rodziny czcionki, np. `Arial`, `Courier`, `Helvetica`, `Times New Roman` itd.
`size=`*`wartość`*	To jest argument w postaci liczby całkowitej określającej w punktach wielkość czcionki.
`weight=`*`wartość`*	Ten argument określa styl czcionki. Dozwolone wartości to `normal` (zwykła) i `bold` (pogrubiona).
`slant=`*`wartość`*	Ten argument określa pochylenie czcionki. Jeżeli chcesz, aby czcionka była pochylona, podaj wartość `italic`. Jeśli czcionka ma nie być pochylona, podaj wartość `roman`.
`underline=`*`wartość`*	Jeżeli tekst ma zostać podkreślony, temu argumentowi przypisz wartość `1`. W przeciwnym razie argument powinien mieć wartość `0`.
`overstrike=`*`wartość`*	Jeżeli tekst ma zostać przekreślony, temu argumentowi przypisz wartość `1`. W przeciwnym razie argument powinien mieć wartość `0`.

Listing 13.22 (font_demo.py)

```
1  # Ten program pokazuje przykład wyświetlenia tekstu w widżecie Canvas.
2  import tkinter
3  import tkinter.font
4
5  class MyGUI:
6      def __init__(self):
7          # Utworzenie okna głównego.
8          self.main_window = tkinter.Tk()
9
10         # Utworzenie widżetu Canvas.
11         self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)
12
13         # Utworzenie obiektu Font.
14         myfont = tkinter.font.Font(family='Helvetica', size=18, weight='bold')
15
16         # Wyświetlenie tekstu.
17         self.canvas.create_text(100, 100, text='Witaj, świecie', font=myfont)
18
19         # Wywołanie metody pack() widżetu Canvas.
20         self.canvas.pack()
21
22         # Wywołanie metody mainloop() modułu tkinter.
23         tkinter.mainloop()
24
25 # Utworzenie egzemplarza klasy MyGUI.
26 my_gui = MyGUI()
```

Rysunek 13.41. Okno wyświetlone przez program z listingu 13.22

Punkt kontrolny

13.19. Jakie współrzędne ma piksel w lewym górnym rogu okna w układzie współrzędnych widżetu `Canvas`?

13.20. Jakie współrzędne ma piksel w prawym dolnym rogu okna o wielkości 640 na 480 pikseli w układzie współrzędnych widżetu `Canvas`?

13.21. Czym się różnią układy współrzędnych widżetu `Canvas` i biblioteki grafiki żółwia?

13.22. Jakie metody widżetu `Canvas` będą używane do narysowania wymienionych tutaj kształtów:
a. okręgu,
b. kwadratu,
c. prostokąta,
d. zamkniętego wielokąta o sześciu bokach,
e. elipsy,
f. wycinka okręgu.

Pytania kontrolne

Test jednokrotnego wyboru

1. Za pomocą _____ użytkownik komunikuje się z komputerem.
   a. procesora
   b. interfejsu użytkownika
   c. systemu sterowania
   d. systemu interaktywnego

2. Zanim spopularyzowane zostały graficzne interfejsy użytkownika, z programów korzystano za pomocą interfejsu _____.
   a. wiersza poleceń
   b. zdalnego terminala

c. sensorycznego
d. sterowanego zdarzeniami

3. _____ to małe okno, w którym można wyświetlić informacje i które umożliwia użytkownikowi wykonanie określonych operacji.
   a. menu
   b. okno zatwierdzające
   c. ekran rozruchowy
   d. okno dialogowe

4. Program _____ jest przykładem programu sterowanego zdarzeniami.
   a. wiersza poleceń
   b. tekstowy
   c. GUI
   d. proceduralny

5. _____ to element będący częścią graficznego interfejsu użytkownika.
   a. gadżet
   b. widżet
   c. narzędzie
   d. obiekt graficzny

6. Tego modułu Pythona można używać do tworzenia programów wyposażonych w graficzny interfejs użytkownika.
   a. `GUI`
   b. `PythonGui`
   c. `tkinter`
   d. `tgui`

7. Ten widżet przedstawia obszar wyświetlający wiersz tekstu.
   e. `Label`
   f. `Entry`
   g. `TextLine`
   h. `Canvas`

8. Ten widżet przedstawia obszar, w którym użytkownik może wprowadzić jeden wiersz tekstu.
   a. `Label`
   b. `Entry`
   c. `TextLine`
   d. `Input`

9. Ten widżet jest kontenerem przechowującym inne widżety.
   a. `Grouper`
   b. `Composer`
   c. `Fence`
   d. `Frame`

10. Ta metoda pozwala na umieszczenie widżetów w odpowiednich położeniach i wyświetla je w oknie głównym programu.
    a. `pack()`
    b. `arrange()`

c. `position()`
d. `show()`

11. _____ to funkcja, która jest wywoływana automatycznie w momencie wystąpienia w programie określonego zdarzenia.
a. funkcja wywołania zwrotnego
b. funkcja automatyczna
c. funkcja rozruchowa
d. wyjątek

12. W tym module została zdefiniowana funkcja `showinfo()`.
a. `tkinter`
b. `tkinfo`
c. `sys`
d. `tkinter.messagebox`

13. Tę metodę wywołujesz, aby zamknąć program wyposażony w GUI.
a. `destroy()` widżetu głównego
b. `cancel()` dowolnego widżetu
c. funkcja `sys.shutdown()`
d. metoda `Tk.shutdown()`

14. Tę metodę wywołujesz, aby pobrać dane z widżetu `Entry`.
a. `get_entry()`
b. `data()`
c. `get()`
d. `retrieve()`

15. Obiekt tego typu może być powiązany z widżetem `Label`, a wszelkie przechowywane w nim dane zostaną wyświetlone w widżecie `Label`.
a. `StringVar`
b. `LabelVar`
c. `LabelValue`
d. `DisplayVar`

16. Jeżeli kontener zawiera grupę tych widżetów, w danej chwili zaznaczony może być tylko jeden z nich.
a. `Checkbutton`
b. `Radiobutton`
c. `Mutualbutton`
d. `Button`

17. Widżet _____ dostarcza metody przeznaczone do rysowania prostych kształtów 2D.
a. `Shape`
b. `Draw`
c. `Palette`
d. `Canvas`

Prawda czy fałsz?

1. Język Python ma wbudowane słowa kluczowe przeznaczone do tworzenia programów wyposażonych w GUI.
2. Każdy widżet ma metodę `quit()`, którą można wywołać, aby zamknąć program.
3. Dane pobierane z widżetu `Entry` zawsze są typu `int`.
4. Związek na wyłączność jest automatycznie tworzony między wszystkimi widżetami `Radiobutton` w tym samym kontenerze.
5. Związek na wyłączność jest automatycznie tworzony między wszystkimi widżetami `Checkbutton` w tym samym kontenerze.

Krótka odpowiedź

1. Co odpowiada za kolejność wykonywania operacji w programie działającym w środowisku tekstowym, takim jak interfejs wiersza poleceń?
2. Do czego służy metoda `pack()` widżetu?
3. Do czego służy funkcja `mainloop()` modułu `tkinter`?
4. Jeżeli utworzysz dwa widżety i wywołasz ich metody `pack()` bez argumentów, jak zostaną one rozmieszczone wewnątrz widżetu nadrzędnego?
5. Jak można wskazać, że widżet powinien zostać wyrównany do lewej strony w widżecie nadrzędnym?
6. Jak można pobrać dane z widżetu `Entry`?
7. Jak można użyć obiektu `StringVar` do uaktualnienia zawartości widżetu `Label`?
8. Jak można użyć obiektu `IntVar` do ustalenia, który widżet `Radiobutton` został zaznaczony w grupie takich widżetów?
9. Jak można użyć obiektu `IntVar` do ustalenia, który widżet `Checkbutton` został zaznaczony?

Warsztat projektanta algorytmów

1. Napisz polecenie tworzące widżet `Label`. Jego elementem nadrzędnym ma być `self.main_window`, natomiast wyświetlanym tekstem `Programowanie daje radość!`.
2. Przyjmuję założenie, że `self.label1` i `self.label2` odwołują się do dwóch widżetów `Label`. Utwórz kod wywołujący metody `pack()` tych widżetów, aby zostały wyrównane do lewej strony w widżecie nadrzędnym.
3. Utwórz polecenie tworzące widżet `Frame`. Jego elementem nadrzędnym powinien być `self.main_window`.
4. Utwórz polecenie wyświetlające informacyjne okno dialogowe wraz z tytułem `Program wstrzymany` i komunikatem `Kliknij OK, aby kontynuować`.

5. Utwórz polecenie tworzące widżet `Button`. Jego elementem nadrzędnym powinien być `self.button_frame`, a tekstem `Oblicz`. Z kolei funkcją wywołania zwrotnego ma być metoda `self.calculate()`.
6. Utwórz polecenie tworzące widżet `Button`, którego kliknięcie spowoduje zamknięcie programu. Jego elementem nadrzędnym powinien być `self.button_frame`, a tekstem `Zakończ`.
7. Przyjmuję założenie, że zmienna `data_entry` odwołuje się do widżetu `Entry`. Utwórz polecenie pobierające dane z widżetu, konwertujące je na typ `int`, a następnie przypisujące zmiennej o nazwie `var`.
8. Przyjmuję założenie, że w programie znajduje się następujące polecenie tworzące widżet `Canvas` i przypisujące go zmiennej `self.canvas`:

   ```
   self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)
   ```

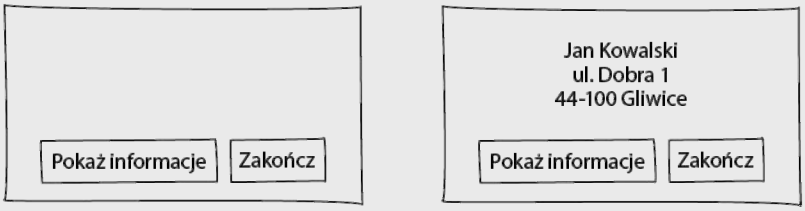
   Utwórz polecenie wykonujące następujące zadania.
   a. Narysowanie niebieskiej linii z lewego górnego do prawego dolnego rogu widżetu `Canvas`. Szerokość linii ma wynosić 3 piksele.
   b. Narysowanie prostokąta z czerwonym obramowaniem i czarnym wypełnieniem. Wierzchołki tego prostokąta mają się znajdować w położeniach określonych współrzędnymi:
      - lewy górny: (50, 50)
      - prawy górny: (100, 50)
      - lewy dolny: (50, 100)
      - prawy dolny: (100, 100)

   c. Narysowanie zielonego okręgu. Punkt środkowy powinien mieć współrzędne (100, 100), a jego promień ma wynosić 50.
   d. Narysowanie wypełnionego kolorem niebieskim wycinka koła, którego lewy górny wierzchołek ma być w punkcie o współrzędnych (20, 20), a prawy dolny (180, 180). Kąt początkowy łuku ma wynosić 0 stopni, a obrót 90 stopni.

Ćwiczenia programistyczne

1. **Dane osobowe**

   Opracuj program wyposażony w GUI, który po naciśnięciu przycisku wyświetli Twoje dane osobowe. Okno programu powinno wyglądać podobnie do okna na szkicu z lewej strony rysunku 13.42. Gdy użytkownik naciśnie przycisk *Pokaż informacje*, program powinien wyświetlić Twoje imię i nazwisko oraz adres zamieszkania — tak jak na szkicu z prawej strony rysunku 13.42.



Rysunek 13.42. Okno programu wyświetlającego dane osobowe

2. **Tłumacz z łaciny**

   Spójrz na przedstawioną tutaj listę łacińskich słów i ich znaczeń w języku polskim.

łacina	polski
sinister	lewy
dexter	prawy
medium	środkowy

   Opracuj program wyposażony w GUI, który będzie tłumaczył słowa z języka łacińskiego na polski. W oknie powinny być umieszczone trzy przyciski, a na każdym z nich powinien widnieć jeden z wyrazów łacińskich. Gdy użytkownik kliknie dany przycisk, program powinien wyświetlić w etykiecie tłumaczenie słowa na język polski.

3. **Zużycie paliwa**

   Opracuj program wyposażony w GUI, który będzie obliczał zużycie paliwa. W oknie programu powinny się znajdować widżety `Entry`, w których użytkownik będzie mógł wprowadzić pojemność baku samochodu (w litrach) oraz liczbę kilometrów, jaką może przejechać samochód na pełnym baku. Po kliknięciu przycisku *Oblicz zużycie paliwa* program powinien wyświetlić liczbę kilometrów, którą może przejechać samochód na 1 litrze paliwa. Skorzystaj z następującego wzoru:

   $$KNL = przejechane\ kilometry / pojemność\ baku$$

4. **Zamiana stopni Celsjusza na stopnie Fahrenheita**

   Opracuj program wyposażony w GUI, który będzie służył do zamiany temperatury wyrażonej w stopniach Celsjusza na stopnie Fahrenheita. Użytkownik powinien mieć możliwość wprowadzenia wartości temperatury w stopniach Celsjusza i kliknięcia przycisku. Program powinien wyświetlić temperaturę w stopniach Fahrenheita. Skorzystaj z następującego wzoru:

   $$F = \frac{9}{5}C + 32$$

   We wzorze F oznacza temperaturę w stopniach Fahrenheita, a C — temperaturę w stopniach Celsjusza.

5. **Podatek od nieruchomości**

   Powiat pobiera podatek od nieruchomości naliczany na podstawie jej szacowanej wartości, wynoszącej 60% faktycznej wartości. Przykładowo jeżeli akr ziemi kosztuje 10000 zł, jego szacowana wartość wyniesie 6000 zł. Podatek od nieruchomości wynosi 75 gr od każdych 100 zł wartości szacowanej. Podatek od akra nieruchomości o wartości szacowanej równej 6000 zł będzie więc wynosił 45 zł. Opracuj program wyposażony w GUI, który po wprowadzeniu przez użytkownika rzeczywistej wartości nieruchomości wyświetli wartość szacowaną i kwotę podatku od nieruchomości.

6. **Warsztat Joe's Automotive**

   Warsztat Joe's Automotive oferuje następujące usługi:
   - wymiana oleju — 30 zł
   - smarowanie — 20 zł
   - sprawdzenie chłodnicy — 40 zł
   - sprawdzenie skrzyni biegów — 100 zł
   - przegląd pojazdu — 35 zł
   - wymiana tłumika — 200 zł
   - wyważanie kół — 20 zł

   Opracuj program wyposażony w GUI wraz z przyciskami opcji pozwalającymi użytkownikowi na wybór dowolnej usługi lub wszystkich usług. Po kliknięciu przycisku program powinien wyświetlić całkowity koszt wybranych usług.

7. **Rozmowy międzymiastowe**

   Operator telekomunikacyjny wprowadził następujące stawki za rozmowy telefoniczne.

Kategoria	Stawka
W ciągu dnia (od 6:00 do 17:59)	7 gr
Wieczorem (od 18:00 do 23:59)	12 gr
Poza szczytem (od północy do 5:59)	5 gr

   Opracuj program wyposażony w GUI pozwalający użytkownikowi na wybór kategorii (przedstawianych przez zbiór przycisków opcji) i podanie w widżecie `Entry` liczby minut rozmów. Informacyjne okno dialogowe powinno wyświetlić kwotę, którą trzeba będzie zapłacić za te rozmowy.

8. **Stary dom**

   Wykorzystaj poznany w rozdziale widżet `Canvas` do narysowania domu. Upewnij się, że umieszczone zostały przynajmniej dwa okna i drzwi. Jeżeli chcesz, możesz narysować także inne obiekty, takie jak niebo, słońce, chmury itd.

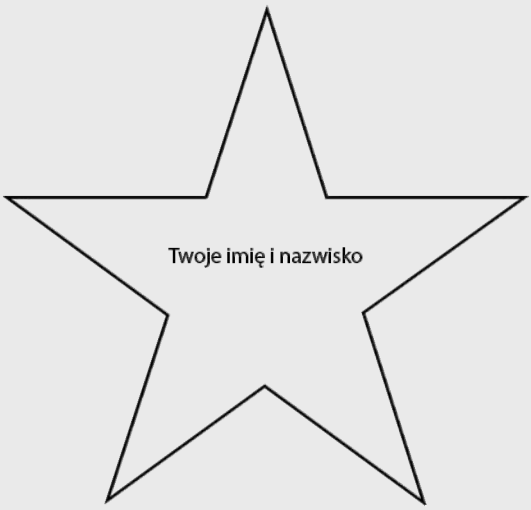
9. **Wiek drzewa**

   Zliczenie pierścieni drzewa to dobry sposób na ustalenie wieku danego drzewa. Każdy pierścień oznacza rok. Wykorzystaj widżet `Canvas` do narysowania

pierścieni pięcioletniego drzewa. Następnie za pomocą metody `create_text()` ponumeruj poszczególne pierścienie, począwszy od środkowego do wysuniętego najbardziej na zewnątrz, i wyświetl powiązany z nimi wiek.

10. **Gwiazda Hollywood**

    Utwórz własną gwiazdę w alei gwiazd Hollywood. Opracuj program wyświetlający gwiazdę podobną do pokazanej na rysunku 13.43 wraz z Twoim imieniem i nazwiskiem.



Rysunek 13.43. Gwiazda Hoolywood

11. **Zarys pojazdu**

    Używając kształtów poznanych w rozdziale, narysuj zarys wybranego pojazdu, np. samochodu, ciężarówki, samolotu itd.

12. **Układ słoneczny**

    Widżet `Canvas` można wykorzystać do narysowania wybranych planet układu słonecznego. Najpierw narysuj Słońce, a następnie poszczególne planety, uwzględniając ich kolejność od Słońca (są to Merkury, Wenus, Ziemia, Mars, Jowisz, Saturn, Uran, Neptun i Pluton). Każdą z nich opisz za pomocą metody `create_text()`.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

>>> PYTHON. LEPSZY NIŻ MYŚLISZ. PRZYDATNIEJSZY NIŻ SĄDZISZ.

Python jest wszechstronnym językiem programowania o imponującej elastyczności i wydajności. Można dzięki niemu rozwiązywać przeróżne problemy programistyczne z różnych dziedzin wiedzy. Nawet jeśli nie masz zamiaru stać się pełnoetatowym programistą, prędko się zorientujesz, jak świetnym i elastycznym narzędziem jest Python. Można za jego pomocą budować oprogramowanie, ale również wspierać pracę statystyków, ekonomistów, maklerów giełdowych, biologów, fizyków czy analityków finansowych. Wystarczy tylko znaleźć sposób analizy problemu i nauczyć się implementować go w programie.

Z tej książki skorzystasz, nawet jeśli nie masz żadnego doświadczenia w projektowaniu i tworzeniu oprogramowania. Dzięki prostym przykładom i zrozumiałemu pseudokodowi, schematom blokowym oraz innym narzędziom zdobędziesz wiedzę o projektowaniu oprogramowania i jego implementowaniu w języku Python. W każdym rozdziale znalazło się wiele przykładowych projektów oraz zwięzłych i praktycznych programów. Książkę rozpoczęto od przedstawienia podstawowych informacji o przechowywaniu danych, danych wejściowych i wyjściowych, struktur kontrolnych, funkcji, sekwencji, list, operacji wejścia-wyjścia oraz obiektów tworzonych za pomocą klas zdefiniowanych w bibliotece standardowej. Następnie omówiono zagadnienia tworzenia klas, dziedziczenia i polimorfizmu, a także definiowania funkcji rekurencyjnych.

W tej książce:

- wprowadzenie do Pythona i środowiska IDLE
- struktury warunkowe, struktury cykliczne i funkcje
- podstawy programowania zorientowanego projektowo
- rekurencja i algorytmy rekurencyjne
- projektowanie GUI aplikacji

Tony Gaddis

od ponad dwudziestu lat prowadzi kursy informatyczne. Najlepiej jest znany w Haywood Community College. Jest cenionym wykładowcą, który ma rzadki dar przekazywania innym wiedzy w łatwy i zajmujący sposób. Dorastał na Hawajach, był surferem i żeglarzem. Studiował grę na gitarze klasycznej i kompozycję muzyki, a później zaczął programować komputery przeznaczone do syntezy muzyki. Obecnie pracuje na Wydziale Informatycznym Florida International University.

ISBN 978-83-283-4682-6

PEARSON
ALWAYS LEARNING

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 99,00 zł