Gniewomir Sarbicki

Python

Kurs dla nauczycieli i studentów

Wydanie II

Helion
Wydawnictwo Naukowe Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/pytkn2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-8239-8

Printed in Poland.

Spis treści

Rozdział 3.

Graficzny interfejs użytkownika

Do zbudowania graficznego interfejsu użytkownika (GUI) można użyć wielu bibliotek, z których najbardziej popularne to GTK, Qt, wx i pythonowy tkinter. Ze względu na prostotę pisania kodu wybierzemy bibliotekę GTK, dostarczaną przez pakiet `gi`. Pakiet ten nie jest częścią biblioteki standardowej i musimy go doinstalować.

3.1 Pierwszy program w GTK

Importujemy moduł `Gtk`:

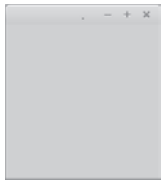
```
1 from gi.repository import Gtk
```

Moduł ten dostarcza nam obiekt `Window`, który reprezentuje główne okno programu. By obiekt z modułu `Gtk` stał się widoczny, wywołujemy jego metodę `show`.

Aplikacje okienkowe cały czas oczekują na zdarzenie wywołane przez użytkownika (przesuwanie i skalowanie okna, naciśnięcie przycisków w oknie, przeciąganie i opuszczanie obiektów itp.). Żeby uruchomić obsługę zdarzeń dla okna, należy wystartować *główną pętlę zdarzeń* poleceniem `Gtk.main()`.

Poniższy program zwróci widoczne okienko programu:

```
1 from gi.repository import Gtk
2
3 o=Gtk.Window()
4 o.show()
5 Gtk.main()
```



Ponieważ okno będzie zawierało wiele innych obiektów `Gtk` jako swoje elementy składowe oraz kilka funkcji obsługujących te obiekty, będziemy cały kod obsługujący okno pakować w klasę. Powyższy kod przyjmie postać:

```
from gi.repository import Gtk


class Okno:

  def __init__(self):
    self.o=Gtk.Window()
    self.o.show()


Okno()
Gtk.main()
```

Wszystkie metody klasy `Gtk.Window` zwróci polecenie `dir(Gtk.Window)`. Do użytku w dalszej części kursu wybierzmy kilka z nich:

- `set_title` — ustawia tytuł okna.
- `move` — ustawia pozycję okna od lewego górnego rogu ekranu wg krotki `(x,y)`.
- `resize` — ustawia rozmiar okna wg krotki `(x,y)`.
- `get_title` — pobiera tytuł okna.
- `get_position` — pobiera bieżącą pozycję okna liczoną od lewego górnego rogu ekranu.
- `get_size` — pobiera bieżący rozmiar okna do krotki.

Działanie powyższych metod ilustruje poniższy kod:

```
from gi.repository import Gtk


class Okno:

  def __init__(self):
    self.o=Gtk.Window()
    self.o.show()


ok=Okno()

ok.o.set_title("Moje okno")
```

```
ok.o.move(500,500)
ok.o.resize(400,400)

input()

print(ok.o.get_title())
print(ok.o.get_position())
print(ok.o.get_size())

Gtk.main()
```

Kod pod nazwą `metody_okna.py` w archiwum `https://ftp.helion.pl/przyklady/pytkn2.zip`.

Pozycja i rozmiar okna uaktualniają się po wystąpieniu jakiegoś zdarzenia, stąd oczekiwanie na naciśnięcie przycisku realizowane przez `input`.

3.2 Umieszczanie w oknie jego obiektów składowych

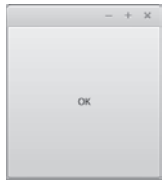
Spośród wielu elementów graficznych, które może zawierać okno aplikacji, wybierzmy cztery:

- `Button` — przycisk. Łańcuch, który stanie się podpisem przycisku, przekazujemy w argumencie nazwanym `label`.
- `TextView` — pole, w którym wyświetlamy tekst.
- `Entry` — jednolinijkowe pole wprowadzania tekstu.
- `Label` — Etykieta tekstowa z nieedytowalnym tekstem, przekazywanym w argumencie nazwanym `label`.

Obiekt dodajemy do okna, podając go jako argument do metody `add` okna. Żeby stał się widoczny, musi być wywołana jego metoda `show`:

```
from gi.repository import Gtk


class Okno:

  def __init__(self):
    self.o=Gtk.Window()
    self.o.set_title('')
    self.b=Gtk.Button(label='OK')
    self.o.add(self.b)
```

```
    self.o.show()
    self.b.show()


ok=Okno()
Gtk.main()
```



Spróbujmy dodać kolejny obiekt. Otrzymamy informację, że okno może zawierać tylko jeden obiekt. Więcej obiektów uzyskujemy, pakując je w pudełkach pionowych: `Gtk.VBox` lub poziomych: `Gtk.HBox`. Mają one metodę `pack_start`, pozwalającą dodawać kolejne elementy do końca pudełka (od dołu dla `VBox` i od prawej dla `HBox`), oraz metodę `pack_end`, pozwalającą dodawać kolejne elementy do początku pudełka (od góry dla `VBox` i od lewej dla `HBox`).

Wywołując metody `pack_start` i `pack_end`, musimy podać dodatkowe argumenty:

- `expand` — jeżeli jest `False`, pakowany obiekt ma przydzieloną minimalną szerokość (dla `HBox`) lub wysokość (dla `VBox`). Dodatkowa przestrzeń jest dzielona po równo pomiędzy elementy, dla których `expand=True`.

- `fill` — czy element powinien wypełniać całą przydzieloną mu przestrzeń. Jeżeli `expand=False`, argument ten nie ma znaczenia.

- `padding` — szerokość dodatkowych marginesów bocznych (dla `HBox`) lub pionowych (dla `VBox`).

Jeżeli oba elementy w pudełku mają wartość `expand=True`, to otrzymujemy:

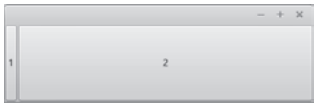
```
from gi.repository import Gtk


class Okno:

  def __init__(self):
    self.o=Gtk.Window()
    self.o.set_title('')
    self.o.resize(400,100)
    self.h=Gtk.HBox()
    self.b=Gtk.Button(label='1')
    self.e=Gtk.Button(label='2')
    self.h.pack_start(self.b,True,True,0)
    self.h.pack_start(self.e,True,True,0)
    self.o.add(self.h)
    self.o.show()
    self.b.show()
    self.e.show()
    self.h.show()
```

```
20
21 ok=Okno()
22 Gtk.main()
```

Kod pod nazwą `gtk_expand.py` w archiwum `https://ftp.helion.pl/przyklady/pytkn2.zip`.

Jeżeli dla jednego elementu `expand=False`, to otrzymujemy:

```
1     self.h.pack_start(self.b,False,True,0)
2     self.h.pack_start(self.e,True,True,0)
```



Jeżeli dla obu elementów `expand=False`, to otrzymujemy:

```
1     self.h.pack_start(self.b,False,True,0)
2     self.h.pack_start(self.e,False,True,0)
```

Obiekty pakowane z `expand=False` zachowają swoją oryginalną szerokość (wysokość dla `VBox`), a dodatkowa przestrzeń zostanie równo podzielona pomiędzy te obiekty, dla których `expand=True`.

Zamiast wywoływać metodę `show` na każdym elemencie okna, można wywołać jego metodę `show_all`.

Ćwiczenie 58. *Stwórz okno pokazane na rysunku obok. Skalowanie pionowe nie powinno zmieniać wysokości dolnego paska, a skalowanie poziome nie powinno zmieniać szerokości przycisku.*

3.3 Obsługa zdarzeń

Aplikacja okienkowa powinna oczekiwać na zdarzenia wywołane przez użytkownika i na nie reagować. W momencie zdarzenia (np. przesunięcia okna, naciśnięcia przycisku) obiekt emituje sygnał informujący o zdarzeniu. Sygnał ten możemy przechwycić dzięki metodzie `connect` tego obiektu, przekazując jej jako argumenty nazwę sygnału i funkcję obsługującą zdarzenie:

```
1   obiekt.connect('nazwa_sygnalu',
      ↪ funkcja_obslugujaca_zdarzenie)
```

Poniżej zostaną opisane możliwe zdarzenia i sygnały, które są wtedy emitowane.

Zamknięcie okna — sygnał `'delete-event'` obiektu `Gtk.Window()`. Sygnał ten jest emitowany, gdy naciśniemy przycisk zamknięcia okna (lub zastosujemy skrót `Alt+F4`). Zauważmy, że gdy zamykamy okno, program dalej działa — nic nie przerywa pętli `Gtk.main()`. Żeby ją przerwać, należy wywołać funkcję `Gtk.` ↪ `main_quit`. Połączmy ją w metodzie `connect` z sygnałem `'delete-event'`, dodając w konstruktorze linię:

```
1     self.o.connect("delete-event",Gtk.main_quit)
```

Teraz zamknięcie okna powoduje również zakończenie programu. Możemy zablokować domyślną obsługę tego zdarzenia (zamknięcia okna), jeżeli funkcja obsługi zdarzenia będzie zwracać `1`:

```
1     self.o.connect("delete-event",lambda *args:1)
```

(Ponieważ nie wiemy, ile argumentów jest przekazywanych do funkcji obsługi zdarzenia, tworzymy funkcję o zmiennej liczbie argumentów). Możemy też stworzyć złośliwy kod:

```
1     self.o.connect("delete-event",lambda *args:[Okno()]*5)
```

Ćwiczenie 59. *Sprawdź, jakie argumenty otrzymuje funkcja obsługi zdarzenia* `'delete-event'`.

Z poziomu programu zamknąć okno możemy wywołując jego metodę `close`.

Naciśnięcie przycisku myszy — sygnał `'button-press-event'`. Sygnał jest emitowany przez elementy składowe okna w momencie naciśnięcia przycisku myszy na ich obszarze. Wypiszmy, jakie argumenty są przekazywane do funkcji obsługi zdarzenia:

```
1 from gi.repository import Gtk
2
```

```
3
4  class Okno:
5
6    def __init__(self):
7      self.o=Gtk.Window()
8      self.o.set_title('')
9      self.o.resize(400,100)
10     self.h=Gtk.HBox()
11     self.b=Gtk.Button(label='Pierwszy')
12     self.e=Gtk.Button(label='Drugi')
13     self.h.pack_start(self.b,True,True,0)
14     self.h.pack_start(self.e,True,True,0)
15     self.o.add(self.h)
16     self.o.show_all()
17     self.b.connect("button-press-event",self.b_press)
18     self.e.connect("button-press-event",self.b_press)
19
20   def b_press(*args):
21     print(args)
22
23
24 ok=Okno()
25 Gtk.main()
```

↓

```
(<__main__.Okno object at 0x7f6d451898d0>, <Button object
    ↪ at 0x7f6d47ffb410 (GtkButton at 0x1845150)>, <void at
    ↪  0x17d7c00>)
```

Kod pod nazwą `gtk_bpe.py` w archiwum `https://ftp.helion.pl/przyklady/pytkn2.zip`.

Pierwszym argumentem jest `self` (pamiętamy, że jako pierwszy argument jest przekazywana instancja wywołująca metodę), drugim jest obiekt, który wyemitował zdarzenie. Trzecim jest obiekt zdarzenia. Żeby sprawdzić, jakie atrybuty ma zdarzenie, przeprowadźmy modyfikację: `print(args)` → `print(dir(args[2]))`. Otrzymamy:

```
['__class__', '__delattr__', '__dict__', '__doc__', '__eq__
    ↪ ', '__format__', '__ge__', '__getattribute__', '
    ↪ __gt__', '__gtype__', '__hash__', '__info__', '
    ↪ __init__', '__le__', '__lt__', '__module__', '__ne__
    ↪ ', '__new__', '__reduce__', '__reduce_ex__', '
    ↪ __repr__', '__setattr__', '__sizeof__', '__str__', '
    ↪ __subclasshook__', '__weakref__', '_get_angle', '
    ↪ _get_center', '_get_distance', 'axes', 'button', '
    ↪ copy', 'device', 'free', 'get', 'get_axis', '
```

```
    ↪ get_button', 'get_click_count', 'get_coords', '
    ↪ get_device', 'get_event_type', 'get_keycode', '
    ↪ get_keyval', 'get_root_coords', 'get_screen', '
    ↪ get_scroll_deltas', 'get_scroll_direction', '
    ↪ get_source_device', 'get_state', 'get_time', '
    ↪ get_window', 'handler_set', 'new', 'peek', 'put', '
    ↪ request_motions', 'send_event', 'set_device', '
    ↪ set_screen', 'set_source_device', 'state', 'time', '
    ↪ triggers_context_menu', 'type', 'window', 'x', '
    ↪ x_root', 'y', 'y_root']
```

W szczególności mamy dostęp do położenia kursora wewnątrz przycisku i na ekranie (`x`, `y`, `x_root`, `y_root`) oraz czasu wystąpienia zdarzenia (`time`).

Możemy każdemu elementowi okna przypisać odrębną funkcję obsługi zdarzenia, ale możemy też w jednej funkcji zróżnicować jej działanie w zależności od jej drugiego argumentu (jego wartością jest element, na którego obszarze nastąpiło zdarzenie).

Ćwiczenie 60. *Stwórz okno z dwoma przyciskami: „OK" oraz „Cancel". Niech naciśnięcie przycisku spowoduje wypisanie jego nazwy w terminalu. Powinno to być zrealizowane za pomocą jednej funkcji obsługi zdarzenia dla obu przycisków.*

Ćwiczenie 61. *Dla najprostszego okna (bez obiektów składowych) dodaj następującą obsługę zdarzenia zamknięcia okna. Przy próbie zamknięcia okna powinno się pojawić okno dialogowe z etykietą „Czy na pewno chcesz zamknąć?" i dwoma przyciskami „OK" i „Cancel" poniżej. Naciśnięcie „OK" powinno kończyć działanie programu, a naciśnięcie „Cancel" tylko zamykać okno dialogowe.*

Zwolnienie przycisku myszy — sygnał `'button-release-event'` emitowany przez elementy okna.

Zmiana położenia lub rozmiaru okna — sygnał `'configure-event'` obiektu `Gtk.Window()`.

Sprawdźmy, jakie argumenty są przekazywane do funkcji:

```
(<__main__.Okno object at 0x7f57ca4368d0>, <Window object
    ↪ at 0x7f57cd263b40 (GtkWindow at 0x182e240)>, <void at
    ↪  0x17f0db0>)
```

Pierwszym argumentem jest instancja, drugim obiekt, który wyemitował sygnał. Trzecim argumentem jest obiekt zdarzenia. Jeżeli wylistujemy go polece-

niem `dir`, zobaczymy atrybuty `x`, `y`, `width`, `height`, w których przechowywane są nowe położenie okna i jego nowy rozmiar.

Ćwiczenie 62. *Stwórz puste okno, które w tytule będzie pokazywać swoje aktualne położenie.*

Ćwiczenie 63. *Stwórz puste okno, które w tytule będzie pokazywać swój aktualny rozmiar.*

Naciśnięcie klawisza — sygnał `'key-press-event'` obiektu `Gtk.Window()`. Jest on emitowany przez okno w momencie, gdy zostaje naciśnięty klawisz i okno jest wtedy aktywne. Znów obiekt zdarzenia jest ostatnim argumentem przekazanym do funkcji obsługi zdarzenia. Wylistowując go, znajdziemy atrybuty o nazwach `hardware_keycode` i `keyval`. Przyjrzyjmy się ich wartościom przy przyciskaniu różnych klawiszy. Wartości tych można używać, by zróżnicować działanie funkcji w reakcji na różne klawisze.

Ćwiczenie 64. *Stwórz puste okno, które reaguje na klawisze:*

- *`strzałka` — przesuwa okno o 10 pikseli we wskazywanym kierunku*
- *`Esc` — kończy program*

Zwolnienie klawisza — sygnał `'key-release-event'` obiektu `Gtk.Window()`.

Ćwiczenie 65. *Rozbuduj program z poprzedniego ćwiczenia, by kombinacja `Shift+strzałka` zmieniała o 10 pikseli rozmiar okna w wybranym kierunku.*

Wejście kursora w obszar okna — sygnał `'enter_notify_event'`.

Ćwiczenie 66. *Stwórz okno uciekające przed kursorem — w funkcji obsługi zdarzenia `'enter_notify_event'` wykryj, z którego boku kursor wszedł na obszar okna, i przesuń okno w przeciwnym kierunku.*

Wyjście kursora z obszaru okna — sygnał `'leave_notify_event'`.

Pełna lista sygnałów GTK jest dostępna pod adresem: `http://lazka.github.io/pgi-docs/Gtk-3.0/classes/Widget.html#signals`.

3.4 Metody elementów okna

W tym podrozdziale poznamy metody pozwalające pobierać tekst z obiektów klasy `Gtk.Entry` oraz zmieniać zawartość obiektu klasy `Gtk.TextView`.

Spośród metod klasy `Gtk.Entry` interesujące dla nas będą dwie:

- `get_text` — pobiera bieżącą zawartość pola.
- `set_text` — ustawia zawartość na podany łańcuch.

W obiekcie klasy `Gtk.TextView` za pomocą metody `set_buffer` ustawiamy bufor tekstowy — obiekt klasy `TextBuffer` — i na nim wykonujemy wszystkie operacje. Dla nas istotna będzie tylko jego metoda `set_text` klasy `Gtk.TextBuffer`, która ustawia zawartość bufora na podany łańcuch.

Ponieważ element `Gtk.TextView` będzie nam służył tylko do wyświetlania wyników, będziemy go ustawiać jako nieedytowalny przy użyciu metody `set_editable` — wywołana z argumentem `False` sprawia, że element jest nieedytowalny (służy tylko do wypisywania tekstu).

Dodatkowo każdy element okna ma metodę `grab_focus` (bez argumentów), która pozwala elementowi przechwycić klawiaturę. Od wywołania tej metody naciśnięcia klawiszy będą kierowane do elementu, który ją wywołał.

Listę wszystkich metod powyższych obiektów uzyskamy, wykonując polecenia: `dir(Gtk.Button)`, `dir(Gtk.TextView)`, `dir(Gtk.TextBuffer)`, `dir(Gtk.Entry)`, lub znajdziemy ją wraz z opisami na stronach:
`https://lazka.github.io/pgi-docs/Gtk-3.0/classes/Button.html`,
`https://lazka.github.io/pgi-docs/Gtk-3.0/classes/TextView.html`,
`https://lazka.github.io/pgi-docs/Gtk-3.0/classes/TextBuffer.html`,
`https://lazka.github.io/pgi-docs/Gtk-3.0/classes/Entry.html`.

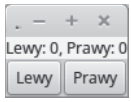
Ćwiczenie 67. *Dla rozwiązania ćwiczenia 58. dodaj obsługę zdarzenia naciśnięcia przycisku „OK”. Po naciśnięciu przycisku bieżąca zawartość pola* `Gtk.Entry` *powinna zostać dodana jako kolejna linia do tekstu w polu* `Gtk.TextView`*, a pole powinno zostać wyczyszczone. Pole* `Gtk.TextView` *powinno być nieedytowalne, a dostęp do klawiatury po uruchomieniu programu powinien należeć do pola* `Gtk.Entry`*.*

Ćwiczenie 68. *Rozbuduj rozwiązanie powyższego ćwiczenia o obsługę klawiszy.* `Enter` *powinien działać tak jak przycisk „OK”.* `Esc` *powinien kończyć program.*

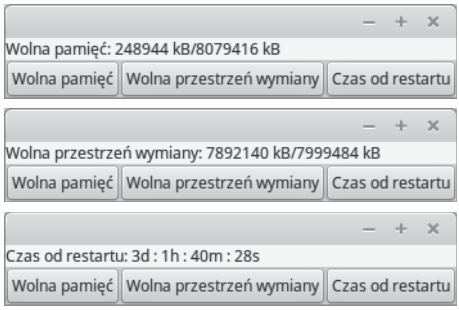
Ćwiczenie 69. *Spraw, by nie dało się przenieść kursora do nieedytowalnego pola* `Gtk.TextView` *oraz by po kliknięciu przycisku „OK" dostęp do klawiatury wracał do pola* `Gtk.Entry`.

Ćwiczenie 70. *Wykorzystując funkcję* `stolica` *z rozwiązania ćwiczenia 38., zmodyfikuj rozwiązanie poprzedniego ćwiczenia tak, by do tekstu w polu* `Gtk.TextView` *dodawana była wartość funkcji* `stolica` *dla argumentu pobranego z* `Gtk.Entry`.

Ćwiczenie 71. *Stwórz okno z nieedytowalnym polem tekstowym i przyciskami „Lewy", „Prawy". Okno powinno zawierać informację o tym, ile razy który przycisk został naciśnięty od początku działania programu, która powinna się uaktualniać przy każdym naciśnięciu przycisku — jak na rysunku obok. Spraw, by nie dało się w w oknie tekstowym umieścić kursora.*



Ćwiczenie 72. *Stwórz okno z nieedytowalnym polem tekstowym i przyciskami jak na rysunku obok. Spraw, by w oknie tekstowym nie dało się umieścić kursora. Okno powinno pokazywać, w zależności od naciśniętego przycisku: ilość wolnej pamięci na całkowitą ilość pamięci, ilość wolnej przestrzeni wymiany na całkowity rozmiar przestrzeni wymiany lub czas od restartu w formacie dni:godziny:minuty:sekundy. Dane dotyczące pamięci można znaleźć w pliku* `/proc/meminfo`. *Czas od restartu (w sekundach) to pierwsza liczba w pliku* `/proc/uptime`.



Dodawanie obrazów do okna*. Obiektem reprezentującym obraz jest `Gtk.Image`. Możemy go bezpośrednio dodać do obiektu okna. Inicjalizujemy go plikiem z obrazem za pomocą jego metody `set_from_file` lub za pomocą metody `set_from_pixbuf` obiektem `Pixbuf` z modułu `GdkPixbuf`, który pozwala na wykonywanie operacji na obrazie.

Obiekt `Pixbuf` tworzymy na podstawie obrazu z pliku za pomocą jego metody `new_from_file`. Możemy go skalować, korzystając z jego metody `scale_simple`, której argumentami są: nowa szerokość, nowa wysokość oraz metoda intepolacji (będziemy wybierać wolną, ale dającą najlepsze rezultaty `GdkPixbuf.InterpType.HYPER`). Z obiektu `Pixbuf` możemy też wyciąć fragment za pomocą jego metody `new_subpixbuf`, której argumentami są cztery

liczby całkowite: współrzędna x lewego górnego rogu wycinka, współrzędna y lewego górnego rogu wycinka, jego szerokość oraz wysokość.

Poniższy program stworzy okno z obrazem, który powstał na skutek przeskalowania obrazu z pliku `test.png` w katalogu roboczym do rozmiaru 500×500, a następnie wycięcia z niego środkowego fragmentu o rozmiarach 300×300:

```
from gi.repository import Gtk, GdkPixbuf


class Okno:

  def __init__(self):
    self.o=Gtk.Window()
    pixbuf = GdkPixbuf.Pixbuf.new_from_file("./test.png")
    pixbuf = pixbuf.scale_simple(500,500,GdkPixbuf.
        ↪ InterpType.HYPER)
    pixbuf = pixbuf.new_subpixbuf(100,100,300,300)
    image = Gtk.Image()
    image.set_from_pixbuf(pixbuf)
    self.o.add(image)
    self.o.show_all()


Okno()
Gtk.main()
```

Kod pod nazwą `gtk_image.py` w archiwum `https://ftp.helion.pl/przyklady/pytkn2.zip`.

Ćwiczenie 73. *Stwórz okno, w którym będzie wyświetlony fragment tapety pulpitu bezpośrednio pod nim. Zakładamy, że znamy zarówno ścieżkę do pliku tapety, jak i wymiary ekranu oraz że obraz z pliku jest skalowany do rozmiarów ekranu.*

Skorowidz

Python — prosto, szybko, skutecznie

- **Konstrukcje języka Python**
- **Sposoby ich użycia**
- **Zadania z rozwiązaniami**

Python to nowoczesny, potężny i uniwersalny język programowania, który zdobył dużą popularność zarówno wśród zawodowców z branży IT, jak i w szkołach i na uczelniach – jako doskonałe narzędzie do nauki programowania na różnych poziomach. Duże możliwości, prostota i zwartość składni, czytelność kodu, wszechstronność zastosowań i wsparcie wielu paradygmatów sprawiają, że co najmniej podstawowa znajomość Pythona stanowi jedno z najczęstszych wymagań wobec poszukujących pracy programistów i inżynierów, nawet jeśli na co dzień mają oni używać zupełnie innych technologii.

To książka pomoże nauczycielom, uczniom, studentom i wszystkim zainteresowanym poznaniem Pythona opanować podstawy tego języka i rozpocząć stosowanie go w praktyce. Krok po kroku uczy czytać i pisać kod, przedstawiając zarówno konstrukcje i funkcje języka, jak i możliwości ich praktycznego użycia do rozwiązywania typowych problemów programistycznych. Programowanie interfejsów graficznych, programowanie wielowątkowe, programowanie sieciowe, tworzenie stron WWW, obliczenia numeryczne – w tym wszystkim świetnie sprawdzi się Python. I wszystko to można znaleźć w tej książce!

- **Pisanie i uruchamianie programów w Pythonie**
- **Podstawowe typy danych i ich zastosowanie**
- **Instrukcje warunkowe, pętle i funkcje**
- **Obsługa wyjątków i używanie plików**
- **Programowanie obiektowe w Pythonie**
- **Graficzny interfejs użytkownika**
- **Wielowątkowość, sieci, bazy danych i strony WWW**
- **Funkcje wyższych rzędów**
- **Iteratory, generatory, koprocedury**
- **Programowanie asynchroniczne i współbieżne**

Programowanie w Pythonie w praktyce

Dr hab. inż. Gniewomir Sarbicki specjalizuje się w informatyce kwantowej. Zawodowo używa Pythona w obliczeniach numerycznych dotyczących splątania kwantowego i jest opiekunem prac inżynierskich powstających z wykorzystaniem tego języka. Od 2012 roku prowadzi kursy w zakresie Pythona na Wydziale Fizyki, Astronomii i Informatyki Stosowanej Uniwersytetu Mikołaja Kopernika w Toruniu.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-8239-8

Cena: 79,00 zł