Fundusze Europejskie dla Wielkopolski
Dofinansowane przez Unię Europejską
SAMORZĄD WOJEWÓDZTWA WIELKOPOLSKIEGO

Programowanie w języku Python

Materiały dydaktyczne dla uczestników kursu

Paweł Sobczak

Konin 2025

Tytuł

Programowanie w języku Python

Materiały dydaktyczne dla uczestników kursu

Autor

Paweł Sobczak

Projekt pn.
„Rozwój studiów o profilu praktycznym i form kształcenia ustawicznego dostosowanych do potrzeb Wielkopolski Wschodniej”,
realizowany przez Akademię Nauk Stosowanych w Koninie,
jest współfinansowany przez Unię Europejską
ze środków Funduszu na rzecz Sprawiedliwiej Transformacji
w ramach programu Fundusze Europejskie dla Wielkopolski 2021-2027

Wydawca
Akademia Nauk Stosowanych w Koninie
ul. Przyjaźni 1, 62-510 Konin

Spis treści

Wprowadzenie

Do nauki programowania można wybrać dowolny język, jednak warto postawić na technologie, które obecnie są wykorzystywane, jest dużo przykładów i materiałów, a sam język ma niski próg wejścia, tzn. już przy podstawowej wiedzy możemy rozpocząć pisanie samodzielnych programów. Takim językiem na pewno jest Python, który jest nowoczesnym, a zarazem uniwersalnym językiem programowania z otwartym dostępem do kodu źródłowego, zawierającym konstrukcje obiektowe, funkcyjne i proceduralne. Jest jednym z najpowszechniej używanych języków programowania na świecie, ma wiele zastosowań, gdyż powstała dla niego olbrzymia liczba bibliotek, które możemy zaimportować do naszego programu. Oprogramowanie napisane w Pythonie działa na większości powszechnie wykorzystywanych platformach (Java, .NET, Android, iOS) i systemach operacyjnych (Unix/Linux, Windows i Mac OS). Python jest bardzo uniwersalnym językiem programowania pozwalającym implantować aplikacje z interfejsem graficznym (z wykorzystaniem bibliotek: pyGTK, pyQt, wxPython, pyKDE, pyGNOME, pyFLTK, FxPy, Tkinter) i tekstowym (curses). Pozwala projektować strony internetowe (Django, Flask, Pyramid, Bottle, Zope2, Web2Py, Web.py) oraz gry (PyGame, pySDL2, Panda3D), a nawet aplikacje mobilne (kivy). Pozwala na analizę tekstu (PLY), obliczenia naukowe (NumPy, SciPy, SimPy), przetwarzanie grafiki (Python Imaging Library (PIL) i pisane skryptów (: Gimp, Blender, VIM, Dia, XUL). Bardzo często jest wykorzystywany do budowania modeli i systemów sztucznej inteligencji (TensorFlow, PyTorch, scikit-learn, Keras, NLTK).

Ze względów objętościowych skupimy się jedynie na podstawowych zagadnieniach z programowania w języku Python, jednak w wystarczającym stopniu, by samodzielnie można było pisać programy. Skrypt należy potraktować jako pewnego rodzaju wprowadzenie w świat programowania, przy minimum teorii i dużej liczbie przykładów i ćwiczeń do samodzielnego rozwiązania. **Aby nauczyć się programowania nie wystarczy czytać, należy pisać i jeszcze raz pisać programy!**

Zanim rozpoczniemy poznawać język i pisać programy, należy przygotować sobie środowisko:

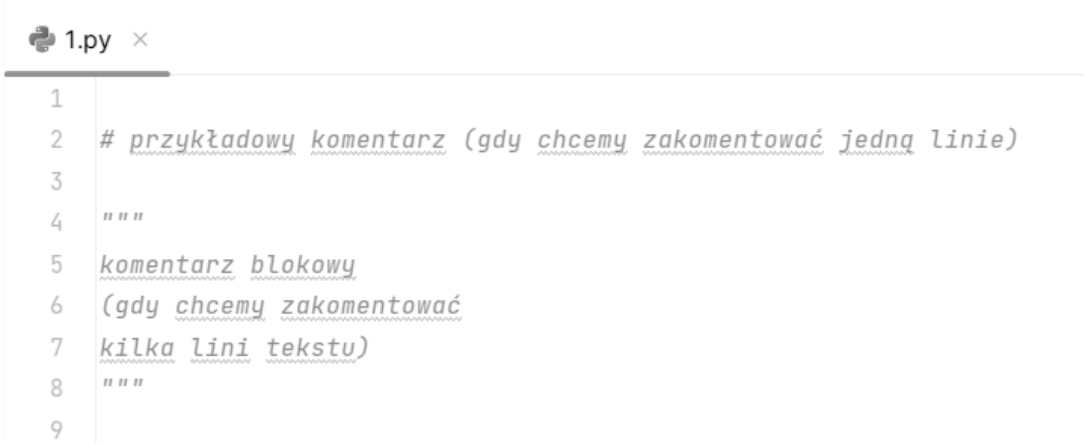
- pobrać aktualną wersję Pythona ze strony https://www.python.org/downloads/
- pobrać środowisko IDE – PyCharm (Community) ze strony, np. dla systemu Windows: https://www.jetbrains.com/pycharm/download/#section=windows

lub wykorzystać edytor i kompilator online, np. https://www.online-python.com/.

Pliki z kodem programu w języku Python zapisuje się z rozszerzeniem *py*, czyli *nazwa.py*. Uruchomienia napisanego kodu programu w *PyCharm* dokonuje za pomocą przycisku *Run* lub poprzez skrót klawiszowy *Shift+F10*.

Komentarz

Bardzo często podczas pisania programów komputerowych komentuje się pewne fragmenty kodu lub pewne linie w celu sprawdzenia innego kodu, gdy nie chcemy usuwać poprzedniej wersji lub chcemy wyjaśnić pewne zapisy w kodzie. Kod programu opatrzony komentarzem nie jest interpretowany przez kompilator. Stosowanie komentarzy w kodzie jest dobrą praktyką i ułatwia zrozumienie kodu nawet po czasie lub przez innego programistę. Wyróżniamy **komentarz jednoliniowy**, jeżeli linie tekstu poprzedzimy znakiem #, oraz **komentarz blokowy**, gdy tekst znajduje się w """ """. Przykład użycia komentarza przedstawia rys. 1.



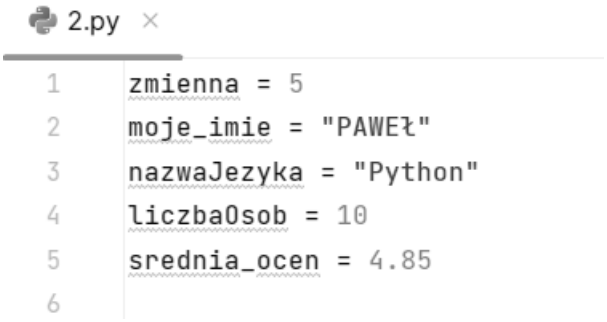
Rys. 1. Komentarz jednoliniowy i blokowy w Pythonie

Źródło: Opracowanie własne.

Zmienne

Pierwszym ważnym elementem w programowaniu są zmienne, czyli nazwy (pudełka), które potrafią przechowywać wartości różnego typu. Operowanie na zmiennych to jedna z najważniejszych funkcjonalności, jakie oferują języki programowania. Krótko można napisać, że zmienna to po prostu nazwa, która wskazuje na jakąś wartość (przechowuje jakąś wartość). Język programowania Python ma kilka wbudowanych typów danych dla zmiennych, takich jak liczby całkowite, rzeczywiste itp. W Pythonie podczas tworzenia (deklarowania)

zmiennej nie trzeba podawać typu danych przechowywanych w zmiennej. Po prostu podaje się nazwę zmiennej i przypisuje się jej wartość, np. liczba_km = 20.5. Znak równości (=) to **operator przypisania**. Na rys. 2. zostały pokazane przykładowe deklaracje zmiennych. Wartości zmiennych mogą się zmieniać w czasie działania programu.



Rys. 2. Deklaracja zmiennych w języku Python

Źródło: Opracowanie własne.

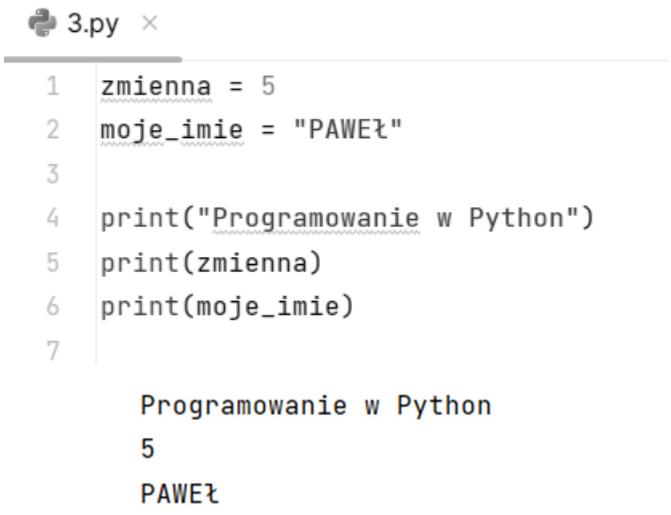
Klika zasad tworzenia zmiennych:

- nazwy zmiennych mogą być dowolnie długie,
- mogą zawierać zarówno litery, jak i liczby, ale nie mogą rozpoczynać się od liczby,
- choć dozwolone jest użycie wielkich liter, w przypadku nazw zmiennych wygodne jest stosowanie wyłącznie małych liter,
- w przypadku nazw złożonych zalecane jest stosowanie znaku podkreślenia _ lub metody camelCase (np. ile_osob_w_sklepie, liczbaOsob),
- takie same nazwy, ale napisane małymi bądź wielkimi literami, oznaczają różne zmienne,
- nazwy zmiennych nie mogą zawierać spacji,
- zmiennym nie można nadawać nazw zastrzeżonych dla instrukcji języka Python (np. and, for, if, del, while, …itp.).

Instrukcja print

Instrukcja *print* pozwala wyświetlać na ekranie tekst lub wartości zmiennych. W ogólności można zapisać *print(wartość)*, gdzie wartością może być tekst ”Programowanie w Python” lub wartość zmiennej. Przykład użycia instrukcji *print* przedstawia rys. 3. W przykładzie zadeklarowano dwie zmienne, w pierwszej kolejności zostaje wyświetlony tekst

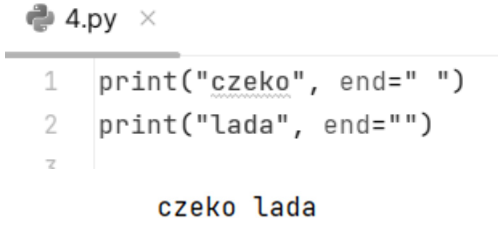
”Programowanie w Python”, a następnie zostają wyświetlone wartości zmiennych *zmienna* i *moje_imie*. W instrukcji *print* tekst zasadniczo powinno się zapisywać w ” ”.



Rys. 3. Instrukcja *print* (kod programu i wynik uruchomienia)

Źródło: Opracowanie własne.

Instrukcja *print* po każdym wyświetleniu wartości (tekstu lub wartości zmiennych) przechodzi do nowej linii. Czasami jednak pożądane jest pozostać w bieżącej linii, wówczas znak nowej linii można zastąpić na inny znak, np. spację: *print(wartość, end=" ")*. Rys. 4 przedstawia przykład zamiany znaku nowej linii na znak spacji w instrukcji *print*. *Print* może również wyświetlić wynik instrukcji (operacji) oraz kilka zmiennych na raz, jak zostało to pokazane na rys. 5. W tym przykładzie pokazany jest wynik odejmowania zmiennych oraz połączenie napisów z wartością zmiennej. Tych zmiennych w instrukcji *print* może być kilka, a nawet kilkanaście. Można również zastosować odpowiednie formatowanie wyświetlanych wartości zmiennych, jednak nie będzie to przedmiotem tego skryptu.



Rys. 4. Instrukcja *print*, zamiana znaku nowej linii na spację

Źródło: Opracowanie własne.

5.py

```
1  zmienna1 = 5
2  zmienna2 = 3
3
4  programista = "Paweł"
5
6  print(zmienna1-zmienna2)
7  print("Witaj ",programista, "!")
```

```
2
Witaj  Paweł !
```

Rys. 5. Wykorzystanie instrukcji *print*

Źródło: Opracowanie własne.

W przypadku zmiennych tekstowych, tzw. łańcuchów, można dokonywać konkatenacji, czyli łączenia kilku zmiennych tekstowych w jeden. Operator + łączy dwa łańcuchy w jedną całość, tworząc nowy, dłuższy łańcuch, jak zostało to pokazane na rys. 6.

6.py

```
1  zm1 = "Witaj "
2  zm2 = "Świecie "
3  zm3 = "!"
4  zm = zm1 + zm2+ zm3
5  print(zm)
```

```
Witaj Świecie !
```

Rys. 6. Konkatenacja łańcuchów

Źródło: Opracowanie własne.

Bardzo często zdarzają się sytuacje, że w jednej instrukcji *print* trzeba wyświetlić zmienne różnych typów, wówczas bardzo pomocna jest notacja *f-string*, która w łatwy sposób pozwala wyświetlić zmienne różnego typu w połączeniu z tekstem. Przykład notacji *f-string* przedstawia rys. 7. Stosując powyższą notację przed cudzysłów zawierający tekst, wstawia się literę *f*, czyli *f"tekst"*, a poszczególne zmienne w tekście umieszcza się w nawiasach klamrowych *{}*, taki zapis w prosty sposób pozwala wyświetlić w instrukcji *print* zmienne różnych typów. Modyfikacji wartości zmiennej wykonujemy podobnie, jak byśmy przypisywali wartość do zmiennej, z tą różnicą, że musimy podać nazwę istniejącej już

zmiennej, aby ją zmodyfikować. Możemy również zmiennej przypisać wartość innej zmiennej, czy też nadpisać wartość innej zmiennej.

nazwaStarejZmiennej = nowaWartośćStarejZmiennej

```
1
2   owoce = "banany"
3   ile_za_kg = 7.66
4   ile_kg = 2
5   wartosc = ile_kg * ile_za_kg
6
7   print(f"Zamówiłem {owoce}, {ile_kg}kg w cenie {ile_za_kg} zł za kilo, do zapłaty {wartosc} zł")
```

```
Zamówiłem banany, 2kg w cenie 7.66zł za kilo, do zapłaty15.32zł
```

Rys. 7. Notacja *f-string*, czyli wyświetlanie zmiennych różnych typów w połączeniu z tekstem

Źródło: Opracowanie własne.

Przykład modyfikacji zmiennej i przypisanie zmiennej wartości innej zmiennej przedstawia rys. 8.

8.py

```
1   a = 12
2   print(a)
3
4   a = 18
5   print(a)
6   b=100
7   b = a
8   print(b)
```

Rys. 8. Zmiana wartości zmiennych

Źródło: Opracowanie własne.

Na ekranie zobaczymy wartości 12, 18, 18, ponieważ na początku zmienna *a* miała wartość 12, następnie wartość zmiennej *a* została zmodyfikowana na wartość 18. Zmienna *b* miała wartość 100, jednak w linii siódmej do *b* została przypisana wartość zmiennej *a*, czyli 18.

Podstawowe typy zmiennych

Jak już zostało wspomniane, język programowania Python nie wymaga podawania typu zmiennej podczas jej deklaracji, jednak możemy wskazać podstawowe typy zmiennych w zależności od przechowywanych przez nie wartości, tj. **liczby** (int, float), **tekst** (str) i **typ logiczny** (bool). W języku tym za pomocą instrukcji *type* można sprawdzić typ zamiennej lub wartości

type(nazwaZmiennej).

Rys. 9 pokazuje, jak sprawdzić typ zmiennej lub wartości. *Int* to zmienne typu całkowitego (pozwalają przechowywać liczby całkowite), *float* to zmienne typu zmiennoprzecinkowego (pozwalają przechowywać liczby rzeczywiste, w których rozdziela się część całkowitą od dziesiętnej za pomocą kropki .). Nic nie stoi na przeszkodzie, by zmienną przekonwertować z typu rzeczywistego na całkowity i odwrotnie. Możemy również liczbę wprowadzoną w postaci napisu przekonwertować na zmienną typu całkowitego lub rzeczywistego, szerzej ten temat będzie poruszony podczas wprowadzania wartości z klawiatury (konsoli). Funkcja *int()* konwertuje zmienną do tupu całkowitego, natomiast funkcja *float()* do typu rzeczywistego. Przykładowe konwersje zmiennych przedstawia rys. 10.

9.py

```
1
2  a = 0.5
3  b = 2
4  c = "Cześć"
5  print (type(a))
6  print (type(b))
7  print (type(c))
8  print (type(100))
9  print (type("Python"))
```

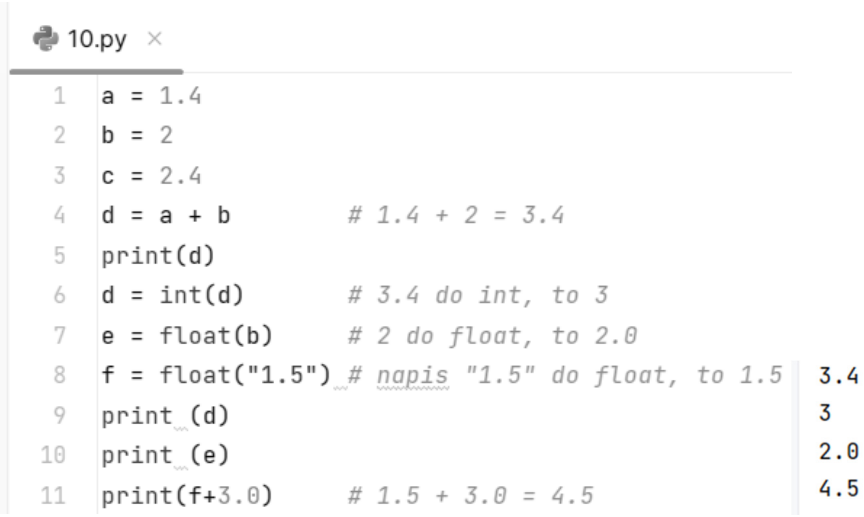
```
<class 'float'>
<class 'int'>
<class 'str'>
<class 'int'>
<class 'str'>
```

Rys. 9. Sprawdzenie typu zmiennej i wartości

Źródło: Opracowanie własne.

Zmienna *d* na początku ma wartość 3.4, jednak po konwersji do typu całkowitego ma wartość 3. Z kolei zmienna *e* to wartość zmiennej *b* przekonwertowana do typu rzeczywistego, czyli 2.0.

Wartość zmiennej *f*, to wartość 1.5, gdyż napis został przekonwertowany do wartości rzeczywistej, a linia 11 spowoduje wyświetlenie wartości 4.5 w konsoli. Język Python bardzo dobrze radzi sobie z konwersją różnych typów zmiennych.



Rys. 10. Konwersja zmiennych

Źródło: Opracowanie własne.

Zmienne tekstowe (str)

Typ *str* reprezentuje w Pythonie wszelkiego rodzaju teksty/napisy. Tekst możemy umieścić pomiędzy znakami pojedynczego (' ') lub podwójnego (" ") cudzysłowu – efekt będzie dokładnie taki sam, np. imie = "Paweł", jezyk = 'Python'. Zadeklarowaliśmy dwie zmienne tekstowe. Również wartości zmiennych typu całkowitego lub rzeczywistego można konwertować do wartości tekstowej za pomocą funkcji *str()*, jak zostało to pokazane na rys. 11. W linii drugiej wiek został skonwertowany do napisu.

Rys. 11. Konwersja zmiennej liczbowej do tekstowej

Źródło: Opracowanie własne.

Zmienne logiczne (bool)

Wartości logiczne reprezentują logiczną prawdę lub fałsz. W języku programowania Python istnieją predefiniowane nazwy odpowiednio *True* dla logicznej prawdy (1) i *False* dla logicznego fałszu (0):

```
p = True      # deklaracja wartości logicznej "prawda"
f = False     # deklaracja wartości logicznej "fałsz".
```

W celu przekształcenia dowolnej wartości na wartość logiczną można wykorzystać funkcję wbudowaną *bool()*. Dla przykładu:

```
a = 1
b = 0
bool(a)      # True
bool(b)      # False.
```

Konwersja wartości zmiennej *a* zwróci *True*, czyli prawdę, a zmiennej *b* zwróci *False*, czyli fałsz.

Operatory arytmetyczne

Żadne obliczenia w programie nie byłyby możliwe, gdyby nie operatory arytmetyczne. Wyróżniamy następujące operatory arytmetyczne (**+**, **-**, *****, **/**, **%**, ******).

Dodawanie **+**	(2 + 4)
Odejmowanie **-**	(11 - 5)
Mnożenie *****	(2 * 4)
Dzielenie **/**	(6 / 2)
Modulo – reszta z dzielenia **%**	(10 % 3)
Potęgowanie ******	(2 ** 2)

Na rys. 12 przedstawiono użycie podstawowych operatorów arytmetycznych (+, -, * i /). Na razie kod programu nie został zabezpieczony na ewentualność dzielenia przez 0,

czyli żeby *liczba2* była różna od zera. Takie zabezpieczenie zrobimy po wprowadzeniu instrukcji warunkowej *if* i operatorów relacji. Warto już zaznaczyć, że w linii 2 i 3 kodu do liczb *liczba1* i *liczba2* została przypisana wartość wczytana z klawiatury, które zostały skonwertowane do typu liczbowego zmiennoprzecinkowego *float*, o czym będzie mowa w dalszej części rozdziału (interaktywny program).

12.py

```
1  print("Prosty kalkulator")
2  liczba1 = float(input("Podaj pierwszą liczbę: "))
3  liczba2 = float(input("Podaj drugą liczbę: "))
4  print("Wynik operacji na liczbach:")
5  print(f"{liczba1} + {liczba2} = {liczba1+liczba2}")
6  print(f"{liczba1} - {liczba2} = {liczba1-liczba2}")
7  print(f"{liczba1} * {liczba2} = {liczba1*liczba2}")
8  print(f"{liczba1} / {liczba2} = {liczba1/liczba2}")
```

```
Prosty kalkulator
Podaj pierwszą liczbę: 3
Podaj drugą liczbę: 2
Wynik operacji na liczbach:
3.0 + 2.0 = 5.0
3.0 - 2.0 = 1.0
3.0 * 2.0 = 6.0
3.0 / 2.0 = 1.5
```

Rys. 12. Operatory arytmetyczne (+, -, *, /)

Źródło: Opracowanie własne.

Operatory relacji

Dostępne operatory relacji, które wykorzystuje się w instrukcji warunkowej *if* oraz pętlach, są zebrane poniżej. Wynikiem porównania jest wartość logiczna *True* dla prawdy lub *False* dla fałszu. Przykłady użycia operatorów relacji zostaną pokazane po wprowadzeniu wspomnianej wyżej instrukcji *if*.

> większy niż
< mniejszy niż
== równy względem
>= większy lub równy względem
<= mniejszy lub równy względem
!= różny względem

Operatory logiczne

W języku programowania Python wykorzystuje się również operatory logiczne, podobnie jak operatory relacji najczęściej w połączeniu z instrukcją warunkową *if* oraz w pętlach. Operator *AND* to w logice „*i*” (koniunkcja), natomiast *OR* to w logice „*lub*” (alternatywa), z kolei *NOT* to

zaprzeczenie (negacja). Gdy zanegujemy prawdę *True*, to otrzymamy fałsz, czyli *False*, i na odwrót. Logiczne „*lub*" zwraca prawdę, gdy przynajmniej jeden z warunków jest prawdziwy, w przeciwnym razie zwraca fałsz. Z kolei logiczne „*i*" zwraca prawdę w przypadku, gdy wszystkie warunki są prawdziwe, w przeciwnym razie zwraca fałsz.

AND

OR

NOT

Interaktywny program

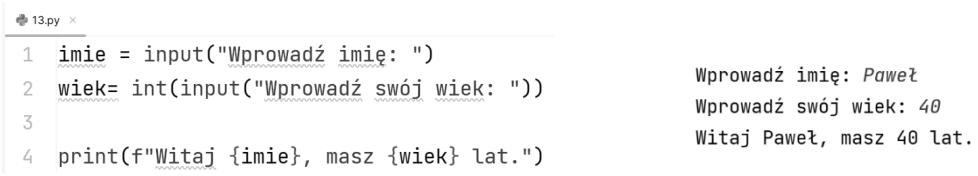
Interaktywny program to program, w którym użytkownik ma możliwość wprowadzania danych w konsoli w czasie rzeczywistym. Python udostępnia wbudowaną funkcję *input*, która umożliwia wprowadzenie danych przez użytkownika. Po wywołaniu funkcji *input* w konsoli pokazuje się kursor oczekiwania na wprowadzenie ciągu znaków. Po wprowadzaniu wartości (ciągu znaków) użytkownik zatwierdza wprowadzoną wartość klawiszem *Enter*. Wprowadzony ciąg znaków z klawiatury musi zostać przypisany do jakieś zmiennej. Warto zaznaczyć bardzo ważną rzecz: jeżeli nawet wczytamy znaki będące liczbą, to należy pamiętać, że to nie jest liczba, tylko ciąg znaków (string), który należy przekonwertować do wartości liczbowej.

imie = input("Wprowadź imię: ")

Po wywołaniu funkcji *input* na ekranie pokazuje się napis *„Wprowadź imię":* oraz kursor oczekiwania na wprowadzenie ciągu znaków. Użytkownik wprowadza imię (ciąg znaków), który zatwierdza klawiszem *Enter*, wprowadzony ciąg znaków zostaje przypisany do zmiennej *imie* (*string* jest zwracany poprzez funkcję *input* i przypisany do zmiennej *imie*). Poniżej podobnie wprowadzany jest ciąg znaków *wiek*, który ma być docelowo wartością liczbową, dlatego dodatkowo jest konwertowany do typu *int*, czyli wartości liczbowej całkowitej. Gdy chcemy dokonać konwersji ciągu znaków do wartości zmiennoprzecinkowej, używamy funkcji *float*.

wiek= int(input("Wprowadź swój wiek: "))

Omawiane wyżej dwa przykłady zostały pokazane rys. 13.



Rys. 13. Interaktywny program, czyli wprowadzanie ciągów znaków z konsoli

Źródło: Opracowanie własne.

Typy sekwencyjne

Sekwencyjne typy danych służą do zapamiętywania wielu wartości w pojedynczej zmiennej, w odróżnieniu od typów prostych, takich jak *int*, *float*, które w pojedynczej zmiennej mogą zachować tylko jedną wartość.

Typ napisowy

Tak naprawdę napisy są sekwencjami znaków. Każdy typ sekwencyjny pozwala na dostęp do każdego swojego elementu z osobna. Aby uzyskać dostęp do znaku na określonej pozycji, podajemy nazwę zmiennej oraz indeks (numer porządkowy liczony od lewej, zero oznacza pierwszy znak napisu) w nawiasach kwadratowych:

imie = ”Paweł”

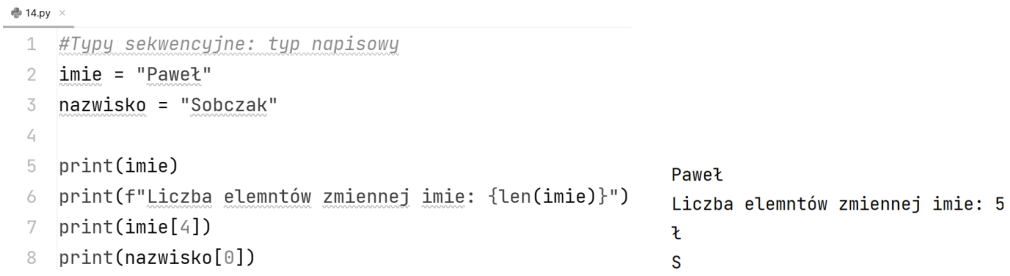
0	1	2	3	4
P	a	w	e	ł

imie[1] *# ‘a’*
print(imie[4]) *# wyświetli znak ł na ekranie konsoli.*

Należy zapamiętać, że znaki (elementy) są numerowane od 0, czyli powyższy napis *imie* składa się z 5 elementów numerowanych od 0 do 4. Można również numerować elementy liczbami ujemnymi, jednak celowo nie wprowadzam tego sposobu indeksowania, by nie zmniejszać czytelności indeksowania typów sekwencyjnych. Aby poznać długość napisu (liczbę elementów), posługujemy się funkcją *len*:

len(imie)* *# 5, numerowane od 0 do 4.

Nic nie stoi na przeszkodzie, by zmienić wartość któregoś z elementów zmiennej napisowej, np. ostatniego *imie[4] = 'l'*. Od tego momentu zamiast znaku *'ł'* będzie znak *'l'*. Przykład odwołania się do pojedynczych znaków napisu oraz długość napisu prezentuje rys. 14.



Rys. 14. Typ napisowy (odwołalnie do pojedynczych znaków, długość napisu)

Źródło: Opracowanie własne.

Dla typu napisowego istnieje szereg przydanych metod, które bardzo ułatwiają pracę programiście, poniżej zostały zaprezentowane przykładowe metody. Jeżeli nasza zmienna napisowa to *imie = "Pawel"*, to możemy dla niej uruchomić pewne metody:

imie.capitalize() – zmienia pierwszą literę na dużą
imie.count(Pa) – zlicza wystąpienie podciągu Pa w napisie imie
imie.isdigit() – sprawdza czy wszystkie znaki są cyframi
imie.islower() – sprawdza czy wszystkie litery są małe
imie.isupper() – sprawdza czy wszystkie litery są duże
imie.replace(old, new) – zastępuje stary podciąg nowym
imie.strip() – usuwa początkowe i końcowe białe znaki.

Warto zaznaczyć, że uruchomienie metody dla zmiennej znakowej odbywa się przez podanie nazwy zmiennej, następnie znaku kroki (.) i nazwy metody. Wynika to z programowania obiektowego, które zostaną przedstawione na szkoleniu.

nazwa_zmiennej.nazwa_metody

Listy

Najpopularniejszym i najczęściej stosowanym typem zmiennych zawierającym więcej niż jedną wartość są listy (od angielskiego „list"), czasami nazywane też tablicami. Lista to

uporządkowany zbiór różnych elementów. Najczęściej wewnątrz listy stosuje się jeden typ zmiennych, ale nic nie stoi na przeszkodzie, aby w jednej liście umieścić wartości zupełnie różnych typów (Python na to pozwala, jednak język C/C++ już nie). Zmienne tworzymy, zapisując pomiędzy nawiasami kwadratowymi („[" i „]") elementy, które chcemy, aby nasza lista przetrzymywała. Poniżej została zadeklarowana *lista1* złożona z 5 elementów numerowanych od 0 do 4 zawierająca zmienne różnego typu.

lista1 = [5, 1.28, "Programowanie", "Python", -2.36]

indeks	0	1	2	3	4
wartość	5	1.28	Programowanie	Python	-2.36

Jak już zostało to wspominane, listy najczęściej są złożone z elementów tego samego typu. Poniżej została zaimplementowana lista złożona z liczb całkowity oraz pusta lista, która nie zawiera na ten moment żadnego elementu.

```
lista1 = [0, 2, 4, 6, 8]
lista2 = []
```

lista1 przechowuje 5 elementów, które zostały umieszczone w nawiasach kwadratowych []. Można wyświetlić wszystkie elementy listy lub pojedynczy element. Można również dokonać zmiany wartości elementu w liście.

```
print(lista1)       # [0, 2, 4, 6, 8]
print(lista1[2])    # 4
lista1[2] = 5       # element o indeksie 2 będzie miał wartość 5
print(lista1)       # [0, 2, 5, 6, 8]
```

Indeksowanie list jest identyczne jak indeksowanie typu napisowego i zaczyna się od 0. Również w innych językach programowania np. C/C++ tablice indeksuje się od 0. Nasza *lista1* składa się z 5 elementów indeksowanych od 0 do 4, gdzie pierwszy element to 0, a ostatni (4) wartość 8. Nie ma indeksu o wartości 5! Jest to bardzo ważne przy pracy z listami, by nie wyjść poza zakres listy.

✓ **Dodawanie elementu do *listy***

Aby dodać element do listy, używamy funkcji *append*, której jako parametr podajemy wartość, jaką chcemy dodać do naszej listy, np.

```
print(lista1)          # [0, 2, 5, 6, 8]
lista1.append(10)      # dodanie elementu 10 do listy
print(lista1)          # [0, 2, 5, 6, 8, 10].
```

Czyli do wyżej omawianej *lista1* został dodany kolejny element o wartości 10. Element ten został dodany na końcu listy.

✓ **Usunięcie elementu po indeksie**

Aby usunąć jakiś element z listy, należy użyć instrukcji *del* od angielskiego słowa *„delete”*, czyli właśnie usuwać. Instrukcja *del* usunie element z listy o określonym indeksie. Należy zwrócić szczególną uwagę na indeks elementu, którego chcemy usunąć, by nie odwoływać się do elementu, który nie istnieje.

```
lista2 = [7, 2, 4, 6, 1]     # [7, 2, 4, 6, 1]
del lista2[2]                # usuniecie elementu z listy o indeksie 2
print(lista2)                # [7, 2, 6, 1]
```

Powyżej został utworzona nowa *lista2* złożona z 5 elementów indeksowana od 0 do 4. Następnie został usunięty element o indeksie numer 2, czyli element **4**, po czym zostały wyświetlone elementy listy. Warto zwrócić uwagę, że po takim działaniu nasza lista uległa skróceniu, a więc i numeracja indeksów uległa zmianie. W tej chwili ostatni element tablicy ma numer 3. Użycie indeksu o wartości 4 spowoduje błąd w programie.

✓ **Dodanie elementu w dowolne miejsce**

Jeśli chcemy dodać element w konkretnym miejscu na liście, musimy znać numer elementu (indeks), przed którym chcemy wstawić nową wartość. Dodanie elementu do listy na konkretnej pozycji wykonuje się przez funkcję *insert*, która przyjmuje dwa parametry. Pierwszy parametr

funkcji *insert* to *indeks* miejsca, przed którym chcemy wstawić nowy element, a drugi to *element*, który chcemy dodać do naszej listy.

```
lista2.insert(2, 4)  # dodanie elementu do listy o wartości 4 na pozycji 2
print(lista2)        # [7, 2, 4, 6, 1]
```

✓ **Sprawdzenie czy element występuje na liście**

W celu sprawdzenia, czy dany element występuje w liście, stosujemy polecenie:

```
lista3 = [0, 2, 4, 6, 8]    # [0, 2, 4, 6, 8]
print(2 in lista3)          # Wyświetli True, bo element 2 jest na liście.
```

Podobnie jak w typie napisowym istnieje kilka przydatnych metod do obsługi list. Przykłady poniżej:

list(s) konwertuje sekwencję s na listę
s.append(x) dodaje nowy element x na końcu listy s
s.count(x) zlicza wystąpienie x w liście s
s.index(x) zwraca najmniejszy indeks i, gdzie s[i] == x
s.pop(i) zwraca i-ty element z listy i usuwa go z listy s
s.remove(x) odnajduje wartość x i usuwa go z listy s
s.reverse() odwraca w miejscu kolejność elementów listy s.

Przykładowe operacje na liście zostały zebrane na rys. 15.

15.py

```
1  lista =["Python", 3, 2.5, "programowanie"]
2  print(lista)                # wyświetlenie elementów listy
3  print(lista[0])             # wyświetlenie elementu o indeksie 0
4  lista1 = [2, 4, 5, 1, 5]
5  print(lista1)
6  print(lista1[4])
7  lista1.append(10)           # dodanie wartości 10 na koniec listy
8  print(lista1)
9  del lista1[1]               # usunięcie elementu o indeksie 1
10 print(lista1)
11 lista1.insert(2,8)          # wstawienie elementu 8 na pozycję 2
12 print(lista1)
13 print(8 in lista1)          # sprawdzenie, czy wartość 8 jest na liście
```

```
['Python', 3, 2.5, 'programowanie']
Python
[2, 4, 5, 1, 5]
5
[2, 4, 5, 1, 5, 10]
[2, 5, 1, 5, 10]
[2, 5, 8, 1, 5, 10]
True
```

Rys. 15. Przykładowe operacje na listach

Źródło: Opracowanie własne.

Krotki

Innym typem zmiennych, który może przetrzymywać więcej niż jedną wartość, są tak zwane krotki. Krotki są bardzo podobne do list z jedną bardzo ważną różnicą – **dane w krotkach są niezmienne**. Czyli raz utworzona krotka już do końca ma takie elementy, jakie zostały podane przy jej implementacji. Krotki deklaruje się tak samo jak listy, tylko zamiast nawiasów kwadratowych używa się zwykłych ().W przypadku krotek, tak samo jak i w listach, możemy wyświetlać pojedyncze elementy za pomocą instrukcji *print* lub wszystkie, również indeksujemy je od zera.

```
krotka1 = (1, 3, 5, 7, 9)
print(krotka1)          # (1, 3, 5, 7, 9)
print(krotka1[1])       # wyświetlamy element 1 czyli 3
```

Krotki są bardzo przydatnym typem danych wszędzie tam, gdzie kolejność elementów ma znaczenie, a bardzo nie chcemy, żeby program mógł zmieniać zawartość naszej zmiennej. Zastosowanie krotki może mieć miejsce w zdefiniowaniu parametrów konfiguracyjnych naszego programu, np. połączenie z bazą danych, login do bazy, hasło itp. Przykład wyświetlenia elementów krotki pokazany jest na rys. 16.

16.py

```
krotka = (3, 2, 1, 5)
krotka1 = ("Ala", "ma", "kota", 3, 3.05)
print(krotka1)
print(krotka)            # wyświetlenie wszystkich elementów
print(krotka[3])         # wyświetlenie pojednyńczego elementu krotki
```

```
('Ala', 'ma', 'kota', 3, 3.05)
(3, 2, 1, 5)
5
```

Rys. 16. Wyświetlanie elementów krotki

Źródło: Opracowanie własne.

Zbiory

Trzecim typem zmiennych mogących posiadać więcej niż jedną wartość są zbiory, mają pewną bardzo ciekawą właściwość, która może być bardzo pomocna przy rozwiązywaniu niektórych problemów: **jej elementy nie mogą się powtarzać**. Dodatkową cechą zbiorów jest to, że są nieuporządkowane, a co za tym idzie, nie możemy wyświetlać dowolnego ich elementu

w taki sposób, jak w przypadku list czy krotek. Można za to dodawać i usuwać elementy ze zbiorów. Zbiory tworzymy za pomocą nawiasów klamrowych {}.

```
zbior1 = {2, 4, 6, 8, 10}      # tworzymy zbiór elementów
print(zbior1)                   # {2, 4, 6, 8, 10}
zbior1.add(1)                   # dodajemy do zbior1 wartość 1
zbior1.add(3) )                 # dodajemy do zbior1 wartość 3
print(zbior1)                   # {1, 2, 3,  4, 6, 8, 10}
zbior.remove(10)                # usuwamy ze zbioru wartość 10
print(zbior)                    # {1, 2, 3,  4, 6, 8}
zbior2 = set()                  # tworzymy zbiór pusty
zbior2= {5, 3, 1}               # przypisujemy lamenty do zbioru
```

Za pomocą metody *add* można dodać elementy do zbioru, z kolei za pomocą *remove* można usunąć element ze zbioru. Funkcja *len* (np. *len*(zbior2)) zwróci liczbę elementów w zbiorze.

Słowniki

Ostatnim z typów danych mogących mieć więcej niż jedną wartość są słowniki. Słowniki są zupełnie innym typem danych od dotychczas opisywanych. Pierwszy element jest nazywany *kluczem*, a drugi *wartością*. Jednemu elementowi (kluczowi) jest przypisana jakaś wartość. Warto zauważyć, że kolejność elementów w słownikach nie ma znaczenia, ponieważ dane w nich i tak wyszukuje się po kluczu. Słowniki można modyfikować, czyli można do nich dodawać nowe elementy i usuwać już istniejące. Jedyna zasada to taka, że **klucze nie mogą się powtarzać**. W słowniku wszystko może być *kluczem*, tak samo jak i wszystko może być *wartością*. Możliwe są zatem takie przykładowe konstrukcje:

```
na_slowa = {1:'jeden', 2:'dwa', 3:'trzy', 4:'cztery', 5:'pięć'}
print(na_slowa)        # {1:'jeden', 2:'dwa', 3:'trzy', 4:'cztery', 5:'pięć'}

na_cyfry = {'jeden':1, 'dwa':2, 'trzy':3, 'cztery':4, 'pięć':5}
print(na_cyfry)        # {'jeden':1, 'dwa':2, 'trzy':3, 'cztery':4, 'pięć':5}
```

Powyższe konstrukcje pozwalają zamieniać napisy na liczby i odwrotnie. Słowniki, tak samo jak zbiory, tworzymy przy użyciu nawiasów klamrowych. Pierwszy element to *klucz*, drugi to *wartość*. *Klucz* jest oddzielony od *wartości* dwukropkiem (:), a poszczególne pary *klucz-wartość* przecinakami (,).

```
print(na_slowa[3])          # wyświetli 'trzy'
print(na_cyfry['trzy'])     # wyświetli 3
```

Aby dodać nowy element do słownika, po prostu wpisujemy w nawiasy kwadratowe klucz, którego chcemy użyć i przypisujemy mu wartość:

```
na_slowa[6]=' sześć'        # dodanie do słownika pary o kluczu 6 i wartości 'sześć'
print(na_slowa)             # {1: 'jeden', 2: 'dwa', 3: 'trzy', 4: 'cztery', 5: 'pięć', 6: 'sześć'}
```

Usuwanie elementów ze słownika wygląda podobnie jak w przypadku list, ale zamiast indeksu elementu podajemy klucz, który chcemy usunąć wykorzystując metodę *pop*.

```
na_slowa.pop(3)         # usunięcie elementu ze słownika na_slowa o kluczu 3
print(na_slowa)         # {1: 'jeden', 2: 'dwa', 4: 'cztery', 5: 'pięć'}
na_cyfry.pop('trzy')    # usunięcie elementu ze słownika na_cyfry o kluczu ‘trzy’
print(na_cyfry)         # {'cztery': 4, 'dwa': 2, 'jeden': 1, 'pięć': 5}
```

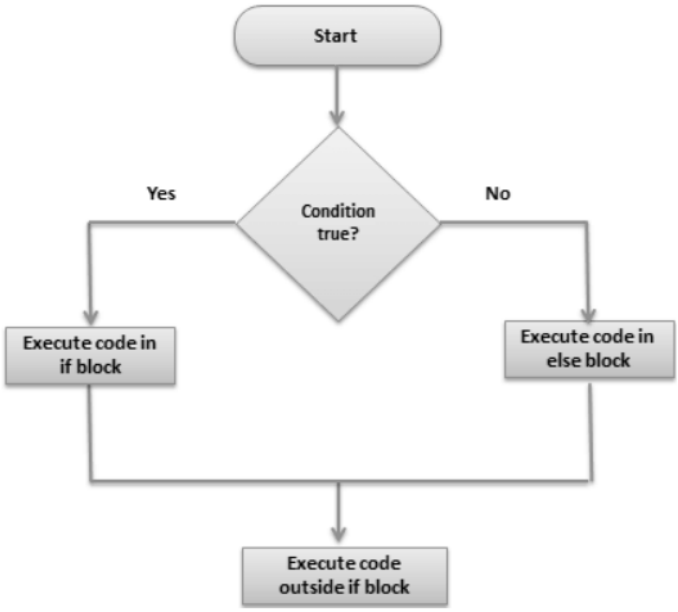
Instrukcje warunkowe if

Praktycznie prawie w każdym programie są podejmowane pewne decyzje, są pewne warunki, które wpływają na pracę programu. Do podejmowania decyzji w programowaniu służy instrukcja warunkowa *if*, czyli w języku angielskim „*jeśli*”. Fragment kodu programu wykona się tylko wtedy, gdy będzie spełniony warunek (warunek będzie prawdziwy, czyli będzie miał wartość logiczną *True*). Sytuacji takich jest bardzo dużo, np. dzielenie zostanie wykonane tylko wtedy, gdy dzielnik będzie różny od zera. Instrukcja *if* posiada również opcjonalną, dodatkową część w postaci instrukcji *else*, czyli „*w przeciwnym wypadku*”. Dodatkowa cześć *else* nie jest obowiązkowa, ale bardzo często jest przydatna, gdy chcemy, by program sprawdził jakiś warunek i wykonał jakiś kod, jeśli warunek jest prawdziwy lub

wykonał inny kawałek kodu, jeśli warunek był nieprawdziwy (fałszywy). Użycie instrukcji *if – else* wygląda w Pythonie następująco:

if *(wyrażenie warunkowe)**:***
    instrukcja 1
    instrukcja 2
    ...
else:
    instrukcja 1
    instrukcja 2
    ...

W części „*wyrażenie warunkowe*" wpisujemy to, co chcemy, aby nasz program sprawdził (czyli stawiamy pewien warunek). Wyrażenie warunkowe może być zapisane w nawiasach, jednak nie jest to wymagane. Po części *wyrażenie warunkowe* musimy wpisać dwukropek, co oznacza, że dalej występują instrukcje, które mają być wykonane, jeśli warunek jest prawdziwy. Warto zaznaczyć, że instrukcji może być dowolna ilość, ale wszystkie instrukcje muszą być wcięte względem instrukcji *if*. W ten sposób Python rozpoznaje, które instrukcje ma wykonać po sprawdzeniu prawdziwości wyrażenia. Tak samo po instrukcji *else* musimy wstawić dwukropek, a instrukcje muszą być wcięte. Np. w języku programowania *C/C++* wcięcia to tylko dobra praktyka programisty, a operacje (instrukcje) blokuje się za pomocą klamer*{}*. Działanie instrukcji *if – else* odzwierciedla rys. 17.



Rys. 17. Instrukcja warunkowa *if – else*

Źródło: Opracowanie własne.

Poniżej prosty przykład użycia instrukcji *if* (rys. 18). Program wczytuje liczbę z konsoli i konwertuje ją do typu całkowitego. Następnie sprawdzany jest warunek, jeśli wprowadzona liczba jest większa od 10, wówczas w konsoli wyświetla się komunikat *'Wpisałeś cyfrę większą niż dziesięć'*, gdy jednak jest mniejsza lub równa 10, to nic się nie dzieje. Kolejną instrukcją jest instrukcja *print* wyświetlająca komunikat *'Koniec programu'* niezależnie od tego, czy warunek był prawdziwy, czy fałszywy.

17.py

```
1  zmienna = int(input('Podaj cyfrę: '))
2  if (zmienna > 10):
3      print('Wpisałeś cyfrę większą niż dziesięć')
4
5  print('Koniec programu')
```

```
Podaj cyfrę: 11
Wpisałeś cyfrę większą niż dziesięć
Koniec programu
```

Rys. 18. Przykład użycia instrukcji warunkowej *if*

Źródło: Opracowanie własne.

Na rys. 19 została pokazana zmodyfikowana wersja programu z rys. 18. Modyfikacja polegała na dodaniu dodatkowej cześci *else*, która wykona się, gdy warunek jest nieprawdziwy, tzn. gdy wprowadzona cyfra jest mniejsza od 10. Program ma jeszcze jeden defekt,

nieprawidłowo się zachowa, gdy wprowadzimy cyfrę równą 10. W celu usunięcia ww. defektu musimy dodać jeszcze jedną instrukcję *if*, jak zostało to pokazane na rys. 20.

18.py

```
1 zmienna = int(input('Podaj cyfrę: '))
2
3 if (zmienna > 10):
4     print('Wpisałeś cyfrę większą niż dziesięć')
5 else:
6     print('Wpisałeś cyfrę mniejszą niż dziesięć')
7
8 print('Koniec programu')
```

```
Podaj cyfrę: 9
Wpisałeś cyfrę mniejszą niż dziesięć
Koniec programu
```

Rys. 19. Przykład użycia instrukcji warunkowej *if – else*

Źródło: Opracowanie własne.

Jak widać na rys. 20, można zagnieżdżać instrukcje warunkowe, czyli w instrukcji warunkowej umieścić kolejną instrukcję. Linie 5 i 6 kodu z rys. 20 można połączyć w jedną linię. Zamiast pisać *else:*, a następnie *if()*, można od razu zapisać *elif*, jak zostało to pokazane na rys. 21.

```
1  zmienna = int(input('Podaj cyfrę: '))
2
3  if (zmienna > 10):
4      print('Wpisałeś cyfrę większą niż dziesięć')
5  else:
6      if (zmienna == 10):
7          print('Wpisałeś cyfrę równą dziesięć')
8      else:
9          print('Wpisałeś cyfrę mniejszą niż dziesięć')
10
11 print('Koniec programu')
```

```
Podaj cyfrę: 10
Wpisałeś cyfrę równą dziesięć
Koniec programu
```

Rys. 20. Przykład użycia zagnieżdżonej instrukcji warunkowej *if – else*

Źródło: Opracowanie własne.

```
1  zmienna = int(input('Podaj cyfrę: '))
2
3  if (zmienna > 10):
4      print('Wpisałeś cyfrę większą niż dziesięć')
5  elif (zmienna == 10):
6          print('Wpisałeś cyfrę równą dziesięć')
7  else:
8          print('Wpisałeś cyfrę mniejszą niż dziesięć')
9
10 print('Koniec programu')
```

```
Podaj cyfrę: 9
Wpisałeś cyfrę mniejszą niż dziesięć
Koniec programu
```

Rys. 21. Przykład użycia instrukcji *elif*

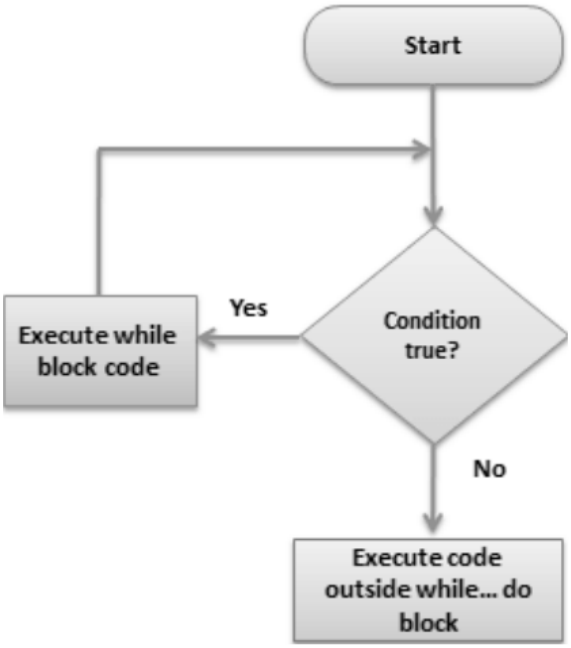
Źródło: Opracowanie własne.

Pętla for

Poznanie instrukcji iteracyjnych (pętli) pozwala programiście rozwiązywać trudniejsze zadania, problemy. Obok instrukcji warunkowej, znajomość pętli jest konieczna do pisania programów. Pisząc programy, bardzo często zdarzy się, że będziemy chcieli wykonać jakieś zadanie (instrukcję) więcej niż jeden raz. Korzystając z pętl, możemy określoną operację (instrukcję) wykonywać z góry określoną liczbę razy, np. 1000, 20000, 3 miliony lub tak długo, jak warunek jest prawdziwy. Ideę działania instrukcji iteracyjnych (pętli) przedstawia rys. 22.

Pierwszą pętlą, którą omówimy, jest pętla *for*. Pętla ta ma kilka postaci. Pierwsza to pętla *for* z zakresem *range*. Ma następującą postać:

***for** i **in range**(101):*
<instrukcje>



Rys. 22. Idea działania pętli

Źródło: Opracowanie własne.

W tej konstrukcji, funkcja *range* przyjmuje parametr *i* zwraca kolejno cyfry z zakresu **<0; wartość)**, czyli w naszym wypadku **<0; 101)**. Pisząc prościej, pętla wykona się 101 razy, a w każdym obiegu pętli parametr (zmienna) *i* będzie przyjmować odpowiednio wartości:

0, 1, 2, 3, 4, …, 100. Warto zwrócić uwagę, jakie wartość przyjmie parametr *i*. Znak mniejszości **<** oznacza, że zaczynamy od 0 włącznie i kończymy na 101 (czyli wartości zapisanej w nawiasie), ale bez tej wartości, bowiem jest nawias otwarty). Po podaniu wartości funkcji *range*() stawia się **:**, a następnie wypisuje się instrukcje, które mają być wykonane w pętli. Wszystkie instrukcje, które mają być wykonywane w pętli, muszą być wcięte w stosunku do instrukcji *for*, podobnie jak to było w przypadki instrukcji *if*. Funkcja *range* może przyjmować następujące postacie:

- ✓ bez zakresu początkowego *range*(10) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
- ✓ z zakresem początkowym i końcowy *range*(1,10) # [1, 2, 3, 4, 5, 6, 7, 8, 9],
- ✓ z zakresem początkowym, końcowym i krokiem *range*(1,10,2) # [1, 3, 5, 7, 9].

23.py

```
1  # Pętla for in range
2  for i in range(10):
3      print(i, end=" ")
4
5  print(" ")
6
7  for i in range(1, 8):
8      print(i, end=" ")
9
10 print(" ")
11
12 for i in range(0, 9, 2):
13     print(i, end=" ")
```

```
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7
0 2 4 6 8
```

Rys. 23. Przykład użycia pętli *for in range*

Źródło: Opracowanie własne.

W ogólności składnia pętli for jest następująca:

for <*nazwa_zmiennej*> ***in*** <*obiekt*> *:*

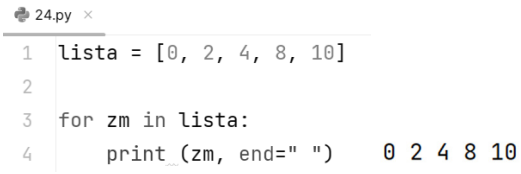
instrukcja1

instrukcja2

instrukcja3

Zamiast <*nazwa_zmiennej*> wstawiamy dowolną nazwę, która będzie wykorzystywana do przechowywania kolejnych elementów pobieranych z <*obiekt*>. Pętle *for* dla przykładu

możemy wykorzystać do wypisania wszystkich elementów z listy. Pętla wykona się tyle razy, ile elementów jest na liście. Przykład użycia pętli *for* do odczytu wszystkich elementów listy przedstawia rys. 24.



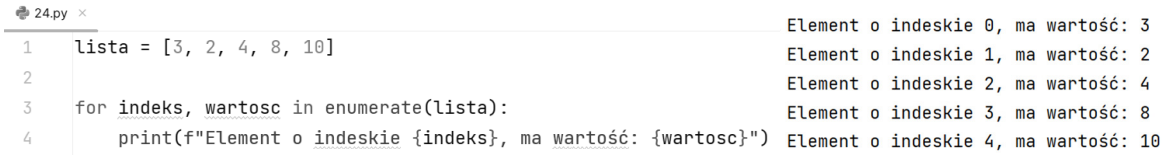
Rys. 24. Przykład użycia pętli *for* z elementami listy

Źródło: Opracowanie własne.

W tym przykładzie pętla *for* pobiera kolejne elementy z listy, „wrzuca" je do zmiennej *zm* i następnie przechodzi do wykonywania instrukcji, które jak zawsze są wypisane po dwukropku i we wcięciu. W tym konkretnym przypadku jest to instrukcja *print*, która wyświetla poszczególne elementy z listy.

Zdarzają się sytuacje, że oprócz wyświetlenia elementów listy potrzebujemy wyświetlić indeksy poszczególnych elementów. W takiej sytuacji stosujemy pętle *for* w postaci *enumerate*:

***for** indeks, wartość in **enumerate**(lista):*
print(indeks, wartość)



Rys. 25. Pętla *for* dla list (wyświetlenie indeksów i wartości elementów listy

Źródło: Opracowanie własne.

Warto przypomnieć, że listy numeruje się od 0. Przykład jak odczytywać indeksy i wartości poszczególnych elementów listy za pomocą instrukcji *for* pokazany jest na rys. 25. Podobnie pętlę *for* można wykorzystać do pracy ze słownikami.

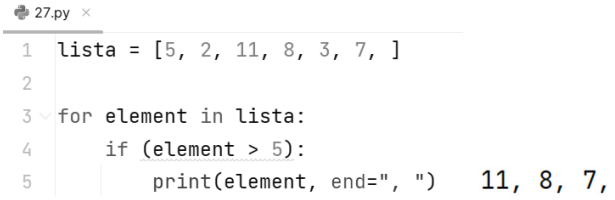
***for** key **in** slownik:*
<instrukcje>

Pętla *for* również pozwala wykonać operacje na słownikach, zawracając jego klucz (*key*). Gdy znamy klucz do słownika, to możemy wyświetlić wartość pod wskazanym kluczem. Przykład wykorzystania pętli *for* do pracy ze słownikami pokazuje rys. 26.

Rys. 26. Przykład użycia pętli *for* dla słowników

Źródło: Opracowanie własne.

Pętle można zagnieżdżać (czyli w pętli może umieszczać kolejne pętle) i łączyć z innymi instrukcjami, np. instrukcją warunkową *if*, jak pokazano na rys. 27.



Rys. 27. Przykład połączenia pętli *for* i instrukcji warunkowej *if*

Źródło: Opracowanie własne.

Powyższy przykład wyświetla elementy listy, jednak tylko te, które są większe od 5.

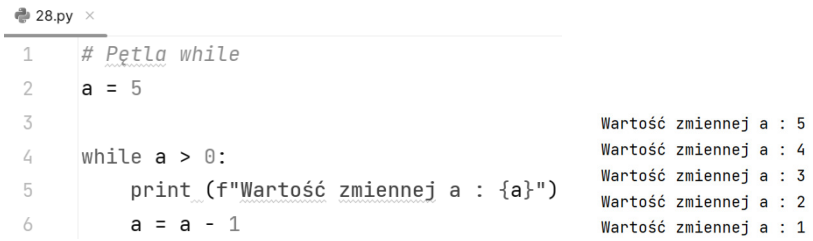
Pętla while

Pęta *for* wykonuje się z góry określoną liczbę razy albo dla wszystkich elementów z listy, słownika, jednak jesteśmy wstanie określić, ile razy ta pętla się wykona. W programowaniu są jednak takie sytuacje, że nie wiemy z góry, ile razy mają się wykonać instrukcje w pętli, wówczas możemy wykorzystać pętlę *while*, która będzie się wykonywać tak długo, jak warunek w niej będzie prawdziwy. Pętla *while*, czyli *„dopóki”*, tak samo jak instrukcja *if*, sprawdza pewien warunek oraz ma podane instrukcje do wykonania. Różnica pomiędzy instrukcją *if* a *while* jest taka, że instrukcja *if* wykona operacje w niej zawarte jeden raz, gdy warunek jest prawdziwy, a pętla *while* tak długo, jak warunek będzie prawdziwy. Czyli pętla *while* sprawdza warunek, wykonuje instrukcje, znowu sprawdza warunek, znowu

wykonuje instrukcje i robi to tak długo, dopóki warunek jest prawdziwy. Jeśli warunek będzie fałszywy (nieprawdziwy) instrukcje nie zostaną wykonane ani razu, a program przejdzie do dalszej części programu. Szkielet pętli *while* wygląda następująco:

while *(warunek):*
instrukcje 1
instrukcje 2

Warto znowu przypomnieć, że instrukcje, które mają być zawarte w pętli muszą być wcięte w stosunku do słowa kluczowego *while*. Przykład użycia pętli *while* jest pokazany na rys. 28.



Rys. 28. Przykład wykorzystania pętli *while*

Źródło: Opracowanie własne.

Na początku zmienna *a* ma wartość 5, ona posłuży do sterowania pętlą, która będzie wykonywała się tak długo, jak wartość zmiennej będzie większa niż 0. Mamy zwarte dwie instrukcje w pętli: wyświetlamy wartość zmiennej i zmniejszamy wartość zmiennej o 1. Omawiając pętle *while* warto wspomnieć o dwóch instrukcjach: *continue* i *break*. Instrukcja *continue* – pomija wykonanie instrukcji i powoduje przejście do kolejnej iteracji (obrotu pętli), z kolei instrukcja *break* – powoduje przerwanie wykonywanie całej pętli. Powyższe instrukcje można stosować zarówno z pętlą *for*, jak i *while*. Rys. 29 pokazuje jak działa instrukcja *continue*, w tym przypadku na ekranie konsoli zostaną wyświetlone liczby 0 i 2 z listy *liczby*. Pozostałe liczby nie zostaną wyświetlone, gdyż operacja *continue* przerywa obieg pętli, gdy liczba *x* będzie mniejsza od 0.

29.py

```
# instrukcja continue
liczby = [-2, -1, 0, -4, 2]
for x in liczby:
    if x < 0:
        continue
    print(x)
```

Rys. 29. Przykład wykorzystania instrukcji *continue*

Źródło: Opracowanie własne.

Na rys. 30 pokazany jest przykład użycia instrukcji *break*, która w tym konkretnym przypadku spowoduje, że w konsoli zostaną wyświetlone jedynie liczby -2 i -1, gdyż *break* przerywa działanie całej pętli, gdy *x* przyjmie wartość równą 0.

30.py

```
#instrukcja break
liczby = [-2, -1, 0, 1, 2]

for x in liczby:
    if x == 0:
        break
    print(x)
```

Rys. 30. Przykład wykorzystania instrukcji *break*

Źródło: Opracowanie własne.

Funkcje w Pythonie

Do tej pory korzystaliśmy tylko z gotowych funkcji (tak na naprawdę metod, ale o tym później), których Python dostarcza olbrzymią ilość. W tym miejscu jednak nauczymy się tworzyć własne funkcje. Jest to bardzo proste i jednocześnie bardzo przyśpiesza tworzenie nowych programów, ponieważ raz napisany kod można bardzo łatwo i szybko wykorzystać ponownie. Nową funkcję deklarujemy używając słowa kluczowego **def** od podania jej nazwy oraz parametrów, jakie funkcja będzie pobierać, jeśli w ogóle jakieś ma pobierać:

def *<nazwa funkcji>():*
instrukcje 1
instrukcje 2

Jeżeli funkcja ma zwracać wartość, to wówczas będzie miała postać:

```
def <nazwa funkcji>():
 instrukcje 1
 instrukcje 2
 return wartość
```

Poniżej mamy przykład funkcji *suma*, która dodaje dwie liczby. Argumentami funkcji są właśnie sumowane liczby (*x*, *y*). Funkcja nic nie zwraca, a jedynie wyświetla wynik sumowania liczb.

```
def suma(x, y):
        z = x + y
        print(z)
```

Tę samą funkcję można również napisać w taki sposób, by zamiast wyświetlania sumy liczb zwracała wynik.

```
def suma(x, y):
        z = x + y
        return z
```

Należy jednak pamiętać, że jeżeli funkcja zwraca wartość, to należy ją przypisać do jakieś zmiennej. Poniżej przykład wywołania funkcji zwracające wartość:

```
a = 3
b = 2
c = suma(a, b)
print(c)
```

Funkcja w tym konkretnym przypadku zwraca wartość 5, gdyż sumuje dwie liczby zapisane w zmiennych *a* i *b* (3, 2). Do zmiennej *c* zostanie przypisana wartość zwracana przez funkcję, a funkcja *print* na ekranie wyświetli wartość 5. Funkcję wywołujemy tak samo jak każdą inną, czyli używając jej nazwy, a w nawiasy wpisując parametry. Parametrami funkcji mogą być zarówno zmienne, jak i stałe. W większości przypadków funkcja zwraca jeden wynik, jednak należy pamiętać, że w Pythonie funkcja może zwrócić ich wiele jednocześnie. Zostało to pokazane na przykładzie poniżej.

```
def licz(x, y):
    z = x - y
    m = x ** y
    r = x + y
    return z, m, r
```

Poniżej wywołanie funkcji z argumentami:

```
wynik = licz(10, 5)
print(wynik[0], wynik[1], wynik[2])
```

Wywołanie funkcji wymaga podania wartości dla wszystkich parametrów – jeżeli nie podamy wartości dla wszystkich parametrów formalnych, wystąpi błąd. Możemy tego uniknąć, podając domyślne wartości argumentów – więcej o tym wątku zostanie powiedziane podczas szkolenia.

Wyjątki

Do tej pory zawsze zakładaliśmy, że nasz kod programu działa poprawnie (np. dzieląc liczby zakładaliśmy, że nikt nie będzie chciał dzielić przez zero). Co się jednak dzieje, kiedy coś pójdzie nie tak? W takich sytuacjach ”wyrzucany” jest wyjątek. Wyjątek jest obiektem specjalnego typu, który powoduje awaryjne przerwanie wykonania programu. Dla przykładu: wyjątek jest ”wyrzucany”, kiedy, na przykład, staramy się odwołać do nieistniejącego elementu w liście lub będziemy próbować dzielić przez zero itd. Jeżeli spróbujemy wykonać kod:

```
lista = [1, 2, 3, 4]
print (lista[5])
```

to wówczas pojawi się wyjątek, jak pokazano na rys. 31.

```
C:\kurs_analiza\Scripts\python.exe C:\kurs_analiza\przy.py
Traceback (most recent call last):
  File "C:\kurs_analiza\przy.py", line 3, in <module>
    print (lista[5])
          ~~~~~^^^
IndexError: list index out of range
```

Rys. 31. Przykład wyjątku podczas odwołania się do nieistniejącego elementu w liście

Źródło: Opracowanie własne.

Jak widać na rys. 31, ostatnia linia informuje, jakiego rodzaju błąd wystąpił. „IndexError" jest typem wyjątku, który oznacza, że numer indeksu, do którego próbujemy się odwołać, jest niepoprawny. Z kolei po dwukropku następuje słowny opis błędu, który w tym przypadku informuje, że podaliśmy zbyt dużą cyfrę jako indeks listy. „Wyrzucony" wyjątek może zostać przez program złapany i obsłużony. Kiedy wyjątek jest „wyrzucany", wykonanie programu jest przerywane i wyjątek jest wyrzucany tak długo, aż zostanie obsłużony lub dopóki nie będzie już nic powyżej, i wtedy program kończy swoje działanie z błędem. Wyjątki obsługuje się specjalną składnią, która wygląda następująco:

```
try:
        instrukcja1
        instrukcja2
        ...
except:
        instrukcja1
        instrukcja2
```

Dla naszego powyższego przykładu, obsługa wyjątku wygadałaby następująco:

```
try:
        a[3]
except:
        print('poza zakresem listy')
```

Taki kod zadziała, „wyrzucany" wyjątek zostanie obsłużony, jednak lepiej zapisać obsługę wyjątku w bardziej ogólnej formie (niezależnie od wartości indeksu i z konkretnym typem wyjątku):

```
lista = [1, 2, 3, 4]
indeks = 5
try:
        print(lista[indeks])
except IndexError:
        print(lista[len(lista)-1])
```

Konstrukcja „*try … except …*" może mieć na końcu dołożoną opcjonalną część *„else"*, która będzie wywoływana, jeśli kod wewnątrz sekkcji „*try"* zostanie wykonany poprawnie bez wyrzucania wyjątku.

```
try:
        print(tablica[indeks])
except IndexError:
        print(tablica[len(tablica)-1])
else:
        print("Kod w bloku try został wykonany poprawnie")
```

Można jeszcze dołożyć jedną opcje (*finally*), która wykona się niezależnie, czy powstanie wyjątek, czy też nie.

```
try:
        print(tablica[indeks])
except IndexError:
        print(tablica[len(tablica)-1])
else:
```

```
        print("Kod w bloku try został wykonany poprawnie")
finally:
        print("Ten print wykona się zawsze bez względu na to czy
        powstanie wyjątek czy nie")
```

Elementy programowania obiektowego

Programowanie obiektowe różni się od tradycyjnego programowania proceduralnego, gdzie dane i procedury nie są ze sobą bezpośrednio związane. Programowanie obiektowe ma ułatwić pisanie, konserwację i wielokrotne użycie programów lub ich fragmentów. W programowaniu obiektowym programista może deklarować własne typy zmiennych, tak zwane *klasy*, które mają w sobie *pola*, czyli *własności*, oraz *zachowanie*, czyli *metody*. Na podstawie wzorca (szablonu), jakim jest *klasa* programista tworzy nowe *obiekty*.

Klasy definiujemy według następującego schematu:

```
class NaszaNowaKlasa:
        pola
        metody
```

Z kolei obiekty tworzymy według następującego schematu:

```
NazwaObiektu = NazwaKlasy(argumenty)
```

Ważnym elementem używania obiektów jest notacja obiektowa. Do pól i metod obiektów dostajemy się, pisząc nazwę zmiennej dowiązanej do obiektu, kropkę i nazwę atrybutu obiektu.

```
nazwaObiektu.nazwaMetody()
nazwaObiektu.nazwaMetody(argumenty)
```

Kolejnym ważnym elementem klasy jest zmienna *self*.
Wewnątrz metod zmienna *self* odnosi się do samego obiektu.

- Dzięki temu możliwy jest dostęp do pól obiektu, np. *self.a.*

- W momencie wywołania metody obiektu, zostaje on automatycznie wstawiony jako pierwszy argument metody i użytkownik podaje o jeden mniej argument niż metoda wymaga.

Przykład klasy przedstawiony jest poniżej:

```
class Wektor():
    def __init__(self, x, y):
        self.a = x
        self.b = y
        print "wektor został stworzony!"

w1 = Wektor(5, 7)      # wektor został stworzony
```

Tak jak zmienna *self* daje dostęp do pól obiektu, tak metoda *__init__* jest wywoływana automatycznie w momencie tworzenia obiektu. Metoda ta powinna wykonywać wszystkie operacje potrzebne do zainicjowania nowego obiektu, w szczególności powinna nadawać wartości jego polom. Oprócz specjalnej metody *__init__* programista może tworzyć własne metody. Metoda jest to funkcja zdefiniowana wewnątrz klasy.

```
class NazwaKlasy:
    def NazwaMetody(self, atrybuty):
        [ciało metody]
```

Wywołanie metody odbywa się następująco:

```
NazwaObiektu = NazwaKlasy(atr1, …, atrN)      # tworzenie obiektu
NazwaObiektu.NazwaMetody(atrybuty)            # wywołanie metody na obiekcie
```

Przykładowe zadanie z rozwiązaniem

Napisz program, który posłuży do przechowania studentów w liście. Utwórz klasę Student:

- Pola: imię, nazwisko, oceny

- Metody: dodajOcene(ocena), wypiszOceny(), policzSrednia()

Utwórz menu: 1 – dodaj studenta, 2 – pokaż studentów, 3 – usuń studenta, 4 – dodaj ocenę studentowi, 5 – wypisz oceny studenta, 6 – średnia studenta, 7 – koniec.

Przykład klasy Student pokazany jest na rys. 32, natomiast dalsza część programu z powyższego zadania przedstawiona jest na rys. 33 i rys. 34.

```
class Student:
    def __init__(self, imie, nazwisko):
        self.imie=imie
        self.nazwisko=nazwisko
        self.oceny=[]

    1 usage
    def dodajOcene(self, ocena):
        self.oceny.append(ocena)

    1 usage
    def wypisz(self):
        for x in self.oceny:
            print(x)

    1 usage
    def policzSrednia(self):
        suma=0
        for x in self.oceny:
            suma=suma+x

        srednia=suma/len(self.oceny)

        print(srednia)
```

Rys. 32. Przykład klasy Student

Źródło: Opracowanie własne.

```
23  listaStudentow = []
24  while True:
25      menu = int(input("1-dodaj studenta, "
26                       "2-pokaz studentow, "
27                       "3-usun studenta, "
28                       "4-dodaj ocene studentowi, "
29                       "5-wypisz oceny studenta, "
30                       "6-srednia studenta, "
31                       "7-koniec"))
32
33      if menu == 1:
34          imie = input("Podaj imie: ")
35          nazwisko = input("Podaj nazwisko: ")
36          student=Student(imie,nazwisko)
37          listaStudentow.append(student)
38
39      elif menu == 2:
40          for x in listaStudentow:
41              print(f"imie: {x.imie}, Nazwisko: {x.nazwisko}")
42
43      elif menu == 3:
44          nazwisko = input("Podaj nazwisko: ")
45          for x in listaStudentow:
46              if x.nazwisko==nazwisko:
47                  listaStudentow.remove(x)
```

Rys. 33. Obsługa programu z klasą Student (cz. 1)

Źródło: Opracowanie własne.

```
48
49      elif menu == 4:
50          nazwisko = input("Podaj nazwisko: ")
51          for x in listaStudentow:
52              if x.nazwisko == nazwisko:
53                  ocena = int(input("Podaj ocena: "))
54                  x.dodajOcene(ocena)
55
56      elif menu == 5:
57          nazwisko = input("Podaj nazwisko: ")
58          for x in listaStudentow:
59              if x.nazwisko == nazwisko:
60                  x.wypisz()
61
62      elif menu == 6:
63          nazwisko = input("Podaj nazwisko: ")
64          for x in listaStudentow:
65              if x.nazwisko == nazwisko:
66                  x.policzSrednia()
67
68      elif menu == 7:
69          break
70
```

Rys. 33. Obsługa programu z klasą Student (cz. 2)

Źródło: Opracowanie własne.

Programowanie obiektowe ma wiele aspektów, które zostaną bardziej szczegółowo omówione podczas szkolenia, min. hermetyzacja, dziedziczenie i wiele innych.

Wprowadzenie do operacji na plikach

Zmienne stanowią pewien sposób przechowywania informacji i uzyskiwania do nich dostępu w trakcie wykonywania programu, jednak po wyłączeniu programu informacje ulatują. Dlatego warto zapisać dane w taki sposób, aby można je było później odzyskać. Do takiego

trwałego zapisu można wykorzystać pliki. Pliki można otworzyć na kilka sposobów (z różnymi atrybutami) w zależności od potrzeby.

- **Odczytywanie danych z plików tekstowych**

```
text_file = open("odczyt.txt", "r")
text_file.close()
```

Powyższe instrukcje pozwalają na otwarcie pliku o nazwie *odczyt.txt* z atrybutem ***r***, czyli do odczytu. Po tej instrukcji możemy czytać dane z pliku. Po zakończeniu czytania należy plik zamknąć. Plik możemy czytać na kilka sposobów, np. linia po linii. Poniżej został przedstawiony przykładowy kod programu, który czyta trzy linie pliku o nazwie *odczyt.txt*, a następnie wyświetla je na ekranie. W tym przykładzie czytamy plik po jednym wierszu naraz. *text_file* to taki uchwyt do pliku.

```
text_file = open("odczyt.txt", "r")
print(text_file.readline())
print(text_file.readline())
print(text_file.readline())
text_file.close()
```

Możemy również wczytać cały pliku do listy, a następnie wyświetlić je linia po linii. Metoda *readline()* czyta jedną linie, z kolei *readlines()* czyta wszystkie linie z pliku.

```
text_file = open("odczyt.txt", "r")
lines = text_file.readlines()
print(lines)            # wszystkie linie od razu
print(len(lines))       # liczba linii
for line in lines:      # tak długo jak są linie
        print(line)     # linia po linii
text_file.close()
```

Tak naprawdę można to zrobić bez instrukcji *readlines()*. Przykład kodu jest zaprezentowany poniżej.

```
text_file = open("odczyt.txt", "r")
for line in text_file:
        print(line)
text_file.close()
```

Wybrane tryby dostępu do pliku tekstowego:

F = open("plik.txt","tryb")

- „*r*" – odczyt danych z pliku tekstowego; jeśli plik nie istnieje, zasygnalizuje błąd,
- „*w*" – zapis danych do pliku tekstowego; jeśli plik już istnieje, jego zawartość zostaje zastąpiona przez nowe dane, jeśli nie istnieje, zostaje utworzony,
- „*a*" – dopisanie danych na końcu pliku tekstowego; jeśli plik istnieje, nowe dane zostają do niego dopisane, jeśli plik nie istnieje, jest tworzony.

- **Zapisywanie łańcuchów znaków do pliku**

```
text_file = open("zapisz.txt", "w")
text_file.write("Wiersz 1\n")
text_file.write("To jest wiersz 2\n")
text_file.write("Ten tekst tworzy wiersz 3\n")
text_file.close()
```

Metoda *write()* zapisuje łańcuch znaków do pliku. Warto wiedzieć, że metoda *write()* nie wstawia automatycznie znaku nowego wiersza na końcu łańcucha, który zapisuje. Należy samemu wstawić znaki nowego wiersza tam, gdzie są potrzebne. Podobnie jak *readlines()*, metoda *writelines()* obsługuje listę łańcuchów, lecz zamiast wczytywać zawartość pliku tekstowego do listy, zapisuje listę łańcuchów do pliku.

```
text_file = open("zapisz_to.txt", "w")
lines = ["Wiersz 1\n",
"To jest wiersz 2\n",
"Ten tekst tworzy wiersz 3\n"]
text_file.writelines(lines)
text_file.close()
```

Wybrane metody do obsługi pliku

- *close()* – zamyka plik; odczytywanie danych z zamkniętego pliku oraz zapisywanie do niego jest niemożliwe, dopóki nie zostanie ponownie otwarty,
- *readline()* – metoda zwraca wszystkie znaki od pozycji bieżącej do końca wiersza,
- *readlines()* – odczytuje wszystkie wiersze pliku i zwraca je jako elementy listy,
- *write(dane)* – zapisuje łańcuch dane do pliku,
- *writelines(dane)* – zapisuje łańcuchy będące elementami listy dane do pliku.

Przykład programu z obsługą plików

Napisz program do wprowadzania studentów w zakresie informacji (imię, nazwisko, grupa). Program ma przechowywać dane w pliku txt, ma umożliwiać dodawanie, usuwanie, zmianę oraz pokazywanie listy studentów. Usuwanie oraz zmianę mogą wykonywać np. po nazwisku. Program wyposażony jest w interaktywne menu (1 – dodaj, 2 – pokaz, 3 – usuń, 4 – zmień, 5 – wyjście).

Przykładowa realizacja zadania z obsługą plików jest pokazana na rys. 34 i rys. 35. W przypadku zapisu do pliku bardziej złożonych informacji, np. list, słowników, a nawet baz danych, służy biblioteka *pickle*. Moduł *pickle* umożliwia „marynowanie" i przechowywanie w pliku bardziej złożonych danych. Przykład marynowania danych zostanie pokazany podczas zajęć.

przy.py

```
while True:
    menu = int(input("1-dodaj, 2-pokaz, 3-usun, 4-zmien, 5-koniec"))

    if menu == 1:
        imie = input("Podaj imie : ")
        nazwisko = input("Podaj nazwisko: ")
        grupa = input("Podaj grupa: ")
        plik=open("85.txt","a")
        plik.write(f"{imie};{nazwisko};{grupa}\n")
        plik.close()

    elif menu == 2:
        plik = open("85.txt", "r")
        for i in plik:
            iSplit=i.split(";")
            print(f"Imie:{iSplit[0]} Nazwisko:{iSplit[1]} Grupa:{iSplit[2]}")
        plik.close()

    elif menu == 3:
        listaTmp =[]
        plik = open("85.txt", "r")
        nazwisko = input("Podaj nazwisko: ")
        for i in plik:
            iSplit = i.split(";")
            if iSplit[1] != nazwisko:
                listaTmp.append(i)
        plik.close()
        plik = open("85.txt", "w")
        plik.writelines(listaTmp)
        plik.close()

```

Rys. 34. Program z obsługą plików (cz. 1)

Źródło: Opracowanie własne.

```
32      elif menu == 4:
33          listaTmp = []
34          plik = open("85.txt", "r")
35          nazwisko = input("Podaj nazwisko: ")
36          for i in plik:
37              iSplit = i.split(";")
38              if iSplit[1] != nazwisko:
39                  listaTmp.append(i)
40              else:
41                  noweImie = input("Podaj nowe imie: ")
42                  noweNazwisko = input("Podaj nowe nazwisko: ")
43                  listaTmp.append(f"{noweImie};{noweNazwisko};{iSplit[2]}")
44          plik.close()
45          plik = open("85.txt", "w")
46          plik.writelines(listaTmp)
47          plik.close()
48
49      elif menu == 5:
50          break
```

Rys. 35. Program z obsługą plików (cz. 2)

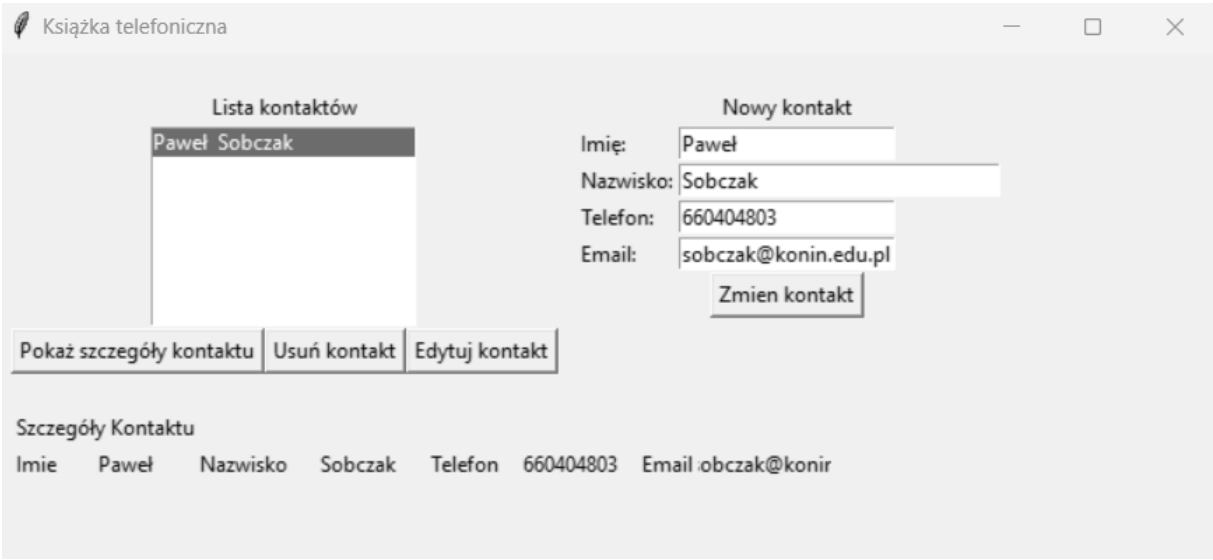
Źródło: Opracowanie własne.

Wprowadzenie do programowania graficznego

Do tej pory wszystkie programy, które pisaliśmy, wykorzystywały do interakcji z użytkownikiem tryb tekstowy (konsolowy). Możemy jednak zaimplementować graficzny interfejs użytkownika (ang. *graphical user interface* – GUI), który udostępnia wizualny sposób interakcji użytkownika z komputerem. Elementy GUI to: ramki, przyciski, pola wejściowe i tekstowe, pola wyboru, przyciski opcji i inne. Do obsługi trybu graficznego można wykorzystać bibliotekę *tkinter*.

```
from tkinter import *
root = Tk()          # okno główne aplikacji
root.mainloop()      # pętla obsługi zdarzeń
```

Przykład programu z interfejsem graficznym jest pokazany na rys. 36.



Rys. 36. Program z interfejsem graficznym

Źródło: Opracowanie własne.

Kod, który realizuje aplikację z rys. 36 został pokazany poniżej. Więcej informacji o tworzeniu aplikacji graficznych w języku Python zostanie przekazanych podczas szkolenia.

```
from tkinter import *

class Kontakt:
    def __init__(self, imie, nazwisko, telefon, email):
        self.imie = imie
        self.nazwisko = nazwisko
        self.telefon = telefon
        self.email = email


listaKontaktow = []

def dodajKontakt():
    imie = entry_Imie.get()
```

```
    nazwisko = entry_Nazwisko.get()
    telefon = entry_Telefon.get()
    email = entry_Email.get()

    kontakt = Kontakt(imie, nazwisko, telefon, email)
    listaKontaktow.append(kontakt)

    entry_Imie.delete(0,END)
    entry_Nazwisko.delete(0,END)
    entry_Telefon.delete(0,END)
    entry_Email.delete(0,END)

    entry_Imie.focus()

    pokazKontakty()


def pokazKontakty():

    listbox_ListaKontaktow.delete(0, END)
    for x, y in enumerate(listaKontaktow):
        listbox_ListaKontaktow.insert(x, f"{y.imie} {y.nazwisko}")


def usunKontakt():
    index = listbox_ListaKontaktow.index(ACTIVE)
    listaKontaktow.pop(index)
    pokazKontakty()

def pokazSzczegoly():
    index = listbox_ListaKontaktow.index(ACTIVE)
    label_SzczegolyKontaktu_ImieV.config(text=listaKontaktow[index].imie)
```

```
    label_SzczegolyKontaktu_NazwiskoV.config(text=listaKontaktow[index].nazwisko)
    label_SzczegolyKontaktu_TelefonV.config(text=listaKontaktow[index].telefon)
    label_SzczegolyKontaktu_EmailV.config(text=listaKontaktow[index].email)


def edytujKontakt():
    index = listbox_ListaKontaktow.index(ACTIVE)

    entry_Imie.delete(0, END)
    entry_Nazwisko.delete(0, END)
    entry_Telefon.delete(0, END)
    entry_Email.delete(0, END)

    entry_Imie.insert(0, listaKontaktow[index].imie)
    entry_Nazwisko.insert(0, listaKontaktow[index].nazwisko)
    entry_Telefon.insert(0, listaKontaktow[index].telefon)
    entry_Email.insert(0, listaKontaktow[index].email)

    button_DodajKontakt.config(text="Zmien kontakt", command=zmienKontakt)

def zmienKontakt():
    # pobierz index zaznaczonego kontaktu
    index = listbox_ListaKontaktow.index(ACTIVE)

    # pobierz z formularza: imie, nazwisko, telefon, mail
    imie = entry_Imie.get()
    nazwisko = entry_Nazwisko.get()
    telefon = entry_Telefon.get()
    email = entry_Email.get()

    # wyedytuj dane obiektu
    listaKontaktow[index].imie = imie
```

```
    listaKontaktow[index].nazwisko = nazwisko
    listaKontaktow[index].telefon = telefon
    listaKontaktow[index].email = email

    # wyczyść pola formularza
    entry_Imie.delete(0, END)
    entry_Nazwisko.delete(0, END)
    entry_Telefon.delete(0, END)
    entry_Email.delete(0, END)

    # ustaw kursow na polu Imię
    entry_Imie.focus()

    # przywróć przycisk do ustawień pierwotnych
    button_DodajKontakt.config(text="Dodaj kontakt", command=dodajKontakt)

    # odśwież listboxa
    pokazKontakty()


root = Tk()
root.geometry("700x300")
root.title("Książka telefoniczna")

ramkaLewa = Frame(root)
ramkaPrawa = Frame(root)
ramkaDolna = Frame(root)

ramkaLewa.grid(row=0, column=0, pady=(20,20), padx=10)
ramkaPrawa.grid(row=0, column=1, sticky=N, pady=(20,20))
ramkaDolna.grid(row=1, column=0, columnspan=2, sticky=W, padx=10)
```

```
label_ListaKontaktow = Label(ramkaLewa, text="Lista kontaktów")
listbox_ListaKontaktow = Listbox(ramkaLewa, width=25, height=7)
button_PokazSzczegoly = Button(ramkaLewa, text="Pokaż szczegóły kontaktu",
command=pokazSzczegoly)
button_UsunKontakt = Button(ramkaLewa, text="Usuń kontakt", command=usunKontakt)
button_EdytujKontakt = Button(ramkaLewa, text="Edytuj kontakt",
command=edytujKontakt)

label_ListaKontaktow.grid(row=0, column=0, columnspan=3)
listbox_ListaKontaktow.grid(row=1, column=0, columnspan=3)
button_PokazSzczegoly.grid(row=2, column=0)
button_UsunKontakt.grid(row=2, column=1)
button_EdytujKontakt.grid(row=2, column=2)

label_NowyKontakt = Label(ramkaPrawa, text="Nowy kontakt")
label_Imie = Label(ramkaPrawa, text="Imię:")
label_Nazwisko = Label(ramkaPrawa, text="Nazwisko:")
label_Telefon = Label(ramkaPrawa, text="Telefon:")
label_Email = Label(ramkaPrawa, text="Email:")
entry_Imie = Entry(ramkaPrawa)
entry_Nazwisko = Entry(ramkaPrawa, width=30)
entry_Telefon = Entry(ramkaPrawa)
entry_Email = Entry(ramkaPrawa)
button_DodajKontakt = Button(ramkaPrawa, text="Dodaj kontakt",
command=dodajKontakt)
# button_DodajKontakt = Button(ramkaPrawa, text="Dodaj kontakt",
command=lambda:test())

label_NowyKontakt.grid(row=0, column=0, columnspan=2)
label_Imie.grid(row=1, column=0, sticky=W)
label_Nazwisko.grid(row=2, column=0, sticky=W)
label_Telefon.grid(row=3, column=0, sticky=W)
```

```
label_Email.grid(row=4, column=0, sticky=W)
entry_Imie.grid(row=1, column=1, sticky=W)
entry_Nazwisko.grid(row=2, column=1, sticky=W)
entry_Telefon.grid(row=3, column=1, sticky=W)
entry_Email.grid(row=4, column=1, sticky=W)
button_DodajKontakt.grid(row=5, column=0, columnspan=2)

label_SzczegolyKontaktu = Label(ramkaDolna, text="Szczegóły Kontaktu")
label_SzczegolyKontaktu_Imie = Label(ramkaDolna, text="Imie")
label_SzczegolyKontaktu_ImieV = Label(ramkaDolna, text="...", width=10)
label_SzczegolyKontaktu_Nazwisko = Label(ramkaDolna, text="Nazwisko")
label_SzczegolyKontaktu_NazwiskoV = Label(ramkaDolna, text="...", width=10)
label_SzczegolyKontaktu_Telefon = Label(ramkaDolna, text="Telefon")
label_SzczegolyKontaktu_TelefonV = Label(ramkaDolna, text="...", width=10)
label_SzczegolyKontaktu_Email = Label(ramkaDolna, text="Email")
label_SzczegolyKontaktu_EmailV = Label(ramkaDolna, text="...", width=10)

label_SzczegolyKontaktu.grid(row=0, column=0, columnspan=8, sticky=W)
label_SzczegolyKontaktu_Imie.grid(row=1, column=0)
label_SzczegolyKontaktu_ImieV.grid(row=1, column=1)
label_SzczegolyKontaktu_Nazwisko.grid(row=1, column=2)
label_SzczegolyKontaktu_NazwiskoV.grid(row=1, column=3)
label_SzczegolyKontaktu_Telefon.grid(row=1, column=4)
label_SzczegolyKontaktu_TelefonV.grid(row=1, column=5)
label_SzczegolyKontaktu_Email.grid(row=1, column=6)
label_SzczegolyKontaktu_EmailV.grid(row=1, column=7)

root.mainloop()
```

Zaprezentowany materiał pozwoli już napisać ciekawe programy, pozostałe zagadnienia zostaną omówione podczas szkolenia.

Zadania

Zadanie 1

Utwórz przykładowy komentarz jednoliniowy i wielowierszowy (blokowy).

Zadanie 2

Utwórz zmienne o dowolnej nazwie, którym przypiszesz wartości: 80, 27.5, Kurs Python.

Zadanie 3

Napisz program, który wykona sumę cen produktów dla konkretnego zamówienia.

Cennik:

- chleb (5,40zł / 1 szt.),
- masło (6,50 zł / 1 szt.),
- pierniki (13,09 / 1 kg),
- sok (4,5 / 1 litr).

Zamówienie: 2 szt. chleba + 3 szt. masła + 1,5 kg pierników + 1 sok.

Zadanie 4

Samochód na 100 km spala 5,2 l paliwa. Ile spali paliwa po przejechaniu 479 km? Wykorzystaj zmienne i operatory w języku Python w celu obliczenia zadania.

Zadanie 5

Napisz program, który prosi użytkownika o podanie imienia, i następnie wypisze na ekran powitanie po imieniu użytkownika, np. *„Witaj Paweł na programowaniu z Pythona”*.

Zadanie 6

Napisz program, który obliczy pole trójkąta na podstawie danych podanych przez użytkownika z konsoli, tj. wysokość (*h*) i długość podstawy tego trójkąta (*a*). Uwzględnij fakt, że wysokość i długość podstawy mogą być liczbami niecałkowitymi. Wzór na obliczeni pola:

$$P_{\Delta} = \frac{1}{2}ah.$$

Zadanie 7

Napisz program obliczający średnią z pięciu liczb podanych przez użytkownika. Liczby mogą być typu zmiennoprzecinkowego.

Zadanie 8

Napisz interaktywny sklep z trzema produktami:

Chleb – 6,50 zł

Sok – 4,00 zł

Pączek – 5,50 zł

Użytkownik będzie pytany o liczbę dla każdej z ww. pozycji asortymentowej, liczba musi być całkowita (*int*). Wypisz podsumowanie zakupów, czyli co zostało kupione, ile sztuk i jaka wartość. Wypisz, ile należy zapłacić całkowicie za złożone zamówienie.

Zadanie 9

Napisz program do nauki tabliczki mnożenia. Program ma wylosować dwie liczby z zakresu (1-10), po czym ma zapytać użytkownika, jaki będzie wynik mnożenia tych liczb. Użytkownik podaje swój wynik. Natomiast po podaniu wyniku przez użytkownika, program wyświetla swój wynik. Na razie nie sprawdzamy, czy wynik podany przez użytkownika jest poprawny. Na przykład:

*Ile to jest 3 * 7 ?*
Odpowiedz użytkownika: 21
Odpowiedź komputera: 21

Zadanie 10

Napisz program, który będzie obliczał potęgę. Potęga zostanie obliczona na podstawie pobranych danych od użytkownika tj. podstawa i wykładnika ($podstawa^{wykladnik}$).

Zadanie 11

Zaprojektuj program, który wczyta od użytkownika dowolny tekst. Program za zadanie wypisać:

- Ile prowadzono znaków,
- Ile jest spacji w wprowadzonym tekście.

Na przykład: „Programowanie w Pythonie”
Liczba znaków: 24
Liczba spacji: 2

Zadanie 12

Napisz program, który 5 razy poprosi o podanie imienia. Podane imiona będą zapisywane do listy. Wypisz dla wszystkich imion z listy poniższy komunikat:
Cześć <tutaj imię z listy>

Zadanie 13

Napisz program, w którym zadeklarujesz dwie listy, które będą przechowywały po 4 dowolne liczby. Na przykład:

lista1 = [1, 3, 2, 5]
lista2 = [4, 5, 1, 8]

Program powinien wyświetlić sumę wszystkich liczb z obu list (29).

Zadanie 14

Napisz program, w którym użytkownik podaje 7 dowolnych liczb całkowitych i dodaje je do listy. Program ma za zadanie:

- wyświetlić wszystkie liczby,
- policzyć sumę wszystkich liczb
- policzyć średnią
- odwrócić kolejność elementów w liście.

Zadanie 15

Napisz program, który 5 razy poprosi użytkownika o wprowadzenie dowolnych liczb całkowitych. Program za zadanie zliczyć, ile wprowadzono liczb unikatowych. Pomocne mogą okazać się zbiory.

Zadanie 16

Wykorzystując poznane typy sekwencyjne, zaprojektuj kod dla poniższej funkcjonalności:
Utwórz zmienne z wartościami:

- zmienna1 = ”jeden”
- zmienna2 = ”pięć”
- zmienna3= ”siedem”

Oblicz sumę ww. zmiennych. Suma zmiennych to 13. W rozwiązaniu problemu pomocne mogą okazać się słowniki.

Zadanie 17

Zaprojektuj program, który dowolną liczbę 4-cyfrową zamieni na interpretację słowną np.

- 9112 # dziewięć jeden jeden dwa
- 5842 # pięć osiem cztery dwa

W rozwiązaniu problemu pomocne mogą okazać się słowniki.

Zadanie 18

Napisz program do nauki tabliczki mnożenia. Program ma wylosować dwie liczby z zakresu (1-10), po czym ma zapytać użytkownika, jaki będzie wynik mnożenia tych liczb. Użytkownik podaje swój wynik. Natomiast po podaniu wyniku przez użytkownika, program wyświetla swój wynik. Program ma również sprawdzić, czy wynik podany przez użytkownika jest poprawny. Na przykład:

*Ile to jest 3 * 7 ?*
Odpowiedź użytkownika: 23
Odpowiedź komputera: 21
Użytkowniku poddałeś błędny wynik!

Zadanie 19

Napisz program, który sprawdza, czy wprowadzona liczba jest liczbą parzystą, czy nieparzystą. Wykorzystaj instrukcję modulo, czyli resztę z dzielenia **%**.

Zadanie 20

Utwórz 2 zmienne, przypisując im dowolne – różne wartości liczbowe. Napisz program, który wskaże największą wartość. Rozszerz program, dodając dodatkową zmienną (trzecią) i przypisz jej dowolną wartość różną od powyższych, który następnie wskaże największą wartość.

Zadanie 21

Utwórz 3 zmienne i przypisz im dowolne wartości liczbowe, np. $a = 10$ $b = 2$ $c = 9$. Wypisz wartości zmiennych od największej do najmniejszej w konsoli.

Zadanie 22

Napisz program, który oblicza wartość współczynnika BMI wg wzoru (*waga* / *wzrost***2). Wzrost podawany jest w metrach. Jeżeli wynik jest w przedziale (18,5–24,9), to wypisuje w konsoli *„waga prawidłowa”*, jeżeli poniżej to *„niedowaga”*, jeżeli powyżej to *„nadwaga”*.

Zadanie 23

Napisz program, który spośród liczb 1-100 wyświetli tylko te, które są podzielne przez 6.

Zadanie 24

Napisz program, który pobiera od użytkownika np. 10 liczb i oblicza sumę tylko tych liczb, które są nieparzyste.

Zadanie 25

Utwórz listę i dodaj do niej w pętli 8 imion. Następnie, wypisz imiona z listy zgodnie z poniższym wzorem: Witaj <imię z listy>.

Zadanie 26

Napisz własny mechanizm obliczania potęgi (bez użycia operatora potęgowania **). Użytkownik podaje podstawę i wykładnik potęgi. W celu napisania powyższego mechanizmu wykorzystaj pętle. Warto jeszcze przypomnieć, że wszystko, co jest podniesione do potęgi 0, jest równe 1, a wykładnikiem potęgi są liczby większe bądź równe 0.

Zadanie 27

Napisz program, który oblicza silnie. Użytkownik podaje dowolną liczbę całkowitą dodatnią, a program zwraca wartość silni z podanej liczby. Np. 5! = 1 * 2 * 3 * 4 * 5 = 120.

Zadanie 28

Napisz grę: komputer losuje liczbę z przedziału 1-100. Użytkownik ma za zadanie odgadnąć, co to za liczba poprzez podawanie kolejnych wartości. Jeżeli podana liczba jest większa od wylosowanej – wyświetlany jest komunikat „podałeś za dużą liczbę", a jeśli mniejsza od wylosowanej – wyświetlony jest komunikat „podałeś za małą liczbę", natomiast gdy podałeś równą wylosowanej – wyświetlony jest komunikat „Gratulacje" i gra zostaje zakończona.

Studium przypadku

Równanie kwadratowe

Napisz program, który rozwiązuje równanie kwadratowe, tzn. oblicza pierwiastki równania kwadratowego. Równanie kwadratowe zapisane w postaci ogólnej wygląda następująco:

$$ax^2 + bx + c = 0,$$

gdzie *a*, *b* i *c* to współczynniki liczbowe i dodatkowo $a \neq 0$ (by równanie było kwadratowe).
W pierwszym kroku oblicza się deltę:

$$\Delta = b^2 - 4ac.$$

Dalsze kroki rozwiązywania równania kwadratowego zależy od wartości Δ:

- ✓ jeżeli $\Delta > 0$, to równanie kwadratowe ma dwa rozwiązania (dwa pierwiastki rzeczywiste)

$$x_1 = \frac{-b-\sqrt{\Delta}}{2a} \qquad x_2 = \frac{-b+\sqrt{\Delta}}{2a},$$

- ✓ jeżeli $\Delta = 0$, to równanie kwadratowe ma jedno rozwiązania (tzw. pierwiastek podwójny)

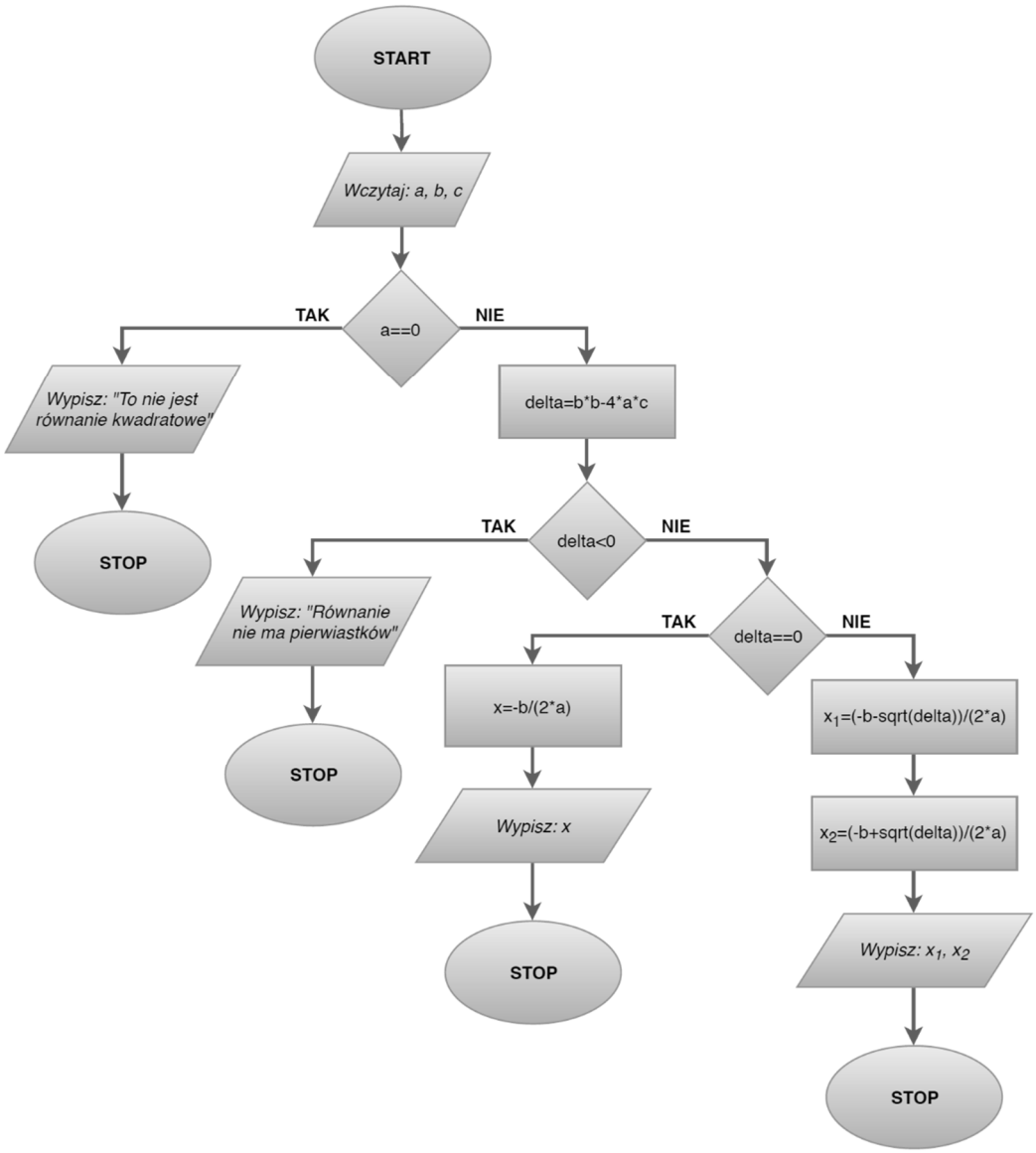
$$x = \frac{-b}{2a},$$

- ✓ jeżeli $\Delta < 0$, to nie istnieją pierwiastki rzeczywiste (równanie posiada jedynie pierwiastki zespolone).

Przepis na rozwiązanie problemu (w naszym przykładzie obliczenie pierwiastków równania kwadratowego) nazywa się algorytmem, który możemy zapisać w postaci słownej lub za pomocą schematów blokowych. Algorytm możemy zapisać również w postaci pseudokodu, więcej na ten temat zostanie przekazane podczas zajęć. Przykładowy schemat blokowy algorytmu rozwiązującego równanie kwadratowe pokazany jest na rys. 37. Opis słowny algorytmu w postaci kroków przedstawiony jest poniżej:

1. START – początek algorytmu
2. Wczytujemy dane wejściowe - współczynniki *a*, *b* i *c* równania kwadratowego
3. Sprawdzamy, czy współczynnik *a* równania jest równy 0
4. Jeżeli $a = 0$, to wypisujemy *„To nie jest równanie kwadratowe”*
5. Jeżeli $a \neq 0$, to obliczamy wartość delty
6. Sprawdzamy, czy Δ jest mniejsza od 0
7. Jeżeli $\Delta < 0$ wypisujemy informację, równanie nie ma pierwiastków (rzeczywistych)
8. Sprawdzamy czy Δ jest równa 0
9. Jeżeli $\Delta = 0$ obliczamy wartość pierwiastka podwójnego *x*
10. Gdy $\Delta = 0$, po obliczeniu *x*, wypisujemy jego wartość
11. Jeżeli $\Delta \neq 0$, a wcześniej Δ nie była mniejsza od 0 (do tego miejsca docieramy, gdy nie został spełniony żaden z poprzednich warunków ($\Delta < 0$ i $\Delta = 0$), czyli gdy $\Delta > 0$. Obliczamy wartości dwóch rozwiązań równania: x_1 i x_2
12. Wypisujemy obliczone wartości obu rozwiązań
13. STOP – wspólny dla wszystkich dróg, koniec algorytmu.

Korzystając ze schematu blokowego lub opisu słownego (listy kroków), możemy przejść do implementacji algorytmu w języku Python. Oczywiście wersji programów może być tyle, ilu jest programistów.



Rys. 37. Przykłady schemat blokowy rozwiązania równania kwadratowego

Źródło: https://stypendium-zawodowe2018.wex.pl/stabilny.html

```
from math import sqrt  #użycie funkcji sqrt z bibl. math
# Program rozwiązuje równie kwadratowe

print("Program rozwiązuje równanie kwadratowe.")
a = float(input("Podaj wartość a: "))
b = float(input("Podaj wartość b: "))
c = float(input("Podaj wartość c: "))

if a == 0:
    print("To nie jest równanie kwadratowe")
```

```
else:
    delta = (b ** 2) - (4 * a * c)

    if delta < 0:
        print("Równanie nie ma pierwiastków (rzeczywistych)")
    elif delta == 0:
        x = -b/(2*a)
        print(f"Wartość pierwiastka podwójnego x = {x}")
    else:
        pdelta = sqrt(delta)
        x1 = (-b - pdelta)/(2*a)
        x2 = (-b + pdelta) / (2 * a)
        print(f"Wartość pierwiastków x1 = {x1}, x2 = {x2}")
```

Rozwiązania zadań

Zadanie 1

```
# przykład użycia instrukcji print
print("Witaj Świecie!")

"""
Programowanie
w
Pytonie
jest
fajne"""
print("Do pracy ...")
```

Zadanie 2

```
wiek = 80
cenaCukierkow = 27.5
kurs_programowania = "Kurs Python."
```

Zadanie 3

```
cenaChleba = 5.40
cenaMasla = 6.50
cenaPierniki = 13.09
cenaSok = 4.5
```

```
zamowienie = 2*cenaChleba + 3*cenaMasla + 1.5*cenaPierniki + 1*cenaSok
print(f"Zamówienie: 2 szt. chleba + 3 szt. masła + 1,5 kg pierników + 1 sok
ma wartość: {zamowienie} zł")
```

Zadanie 4

```
ile_na_100 = 5.2
droga = 479
spalanie = (droga/100)*ile_na_100
print(f"Po przejechaniu {droga} km samochód spali {spalanie} litrów
paliwa.")
```

Zadanie 5

```
imie = input("Podaj imie: ")
print(f"Witaj {imie} na programowaniu z Pythona.")
```

Zadanie 6

```
print("Program oblicza pole trójkąta!")
h = float(input("Podaj wyskość trójkąta: "))
a = float(input("Podaj długość podstawy trójkąta: "))
pole = 0.5*a*h
print(f"Pole trójkąta o wykości {h} i długości podstawy {a} wynosi: {pole} ")
```

Zadanie 7

```
print("Program oblicza średnią z pięciu zadeklarownych liczb.")
l1 = 1.0
l2 = 2.0
l3 = 3.0
l4 = 4.0
l5 = 5.0
suma = l1 + l2 + l3 + l4 + l5
srednia = suma/5.0
print(f"Średnia z liczb: {l1}, {l2}, {l3}, {l4}, {l5}, to: {srednia}")
```

Zadanie 8

```
print("Sklep")
chleb = 6.50
sok = 4.00
paczek = 5.50
```

```
ile_chlebow = int(input("Ile sztuk chleba chcesz zamówić? "))
ile_sokow = int(input("Ile sztuk soków chcesz zamówić? "))
ile_paczkow = int(input("Ile pączków chcesz zamówić? "))
wartosc_chleb = ile_chlebow * chleb
wartosc_sok = ile_sokow * sok
wartosc_paczkow = ile_paczkow * paczek
print("")
print(f"Zamówiłeś {ile_chlebow} chlebów o wartości: {wartosc_chleb}")
print(f"Zamówiłeś {ile_sokow} soków o wartości: {wartosc_sok}")
print(f"Zamówiłeś {ile_paczkow} pączków o wartości: {wartosc_paczkow}")
zamownienie = wartosc_chleb + wartosc_sok + wartosc_paczkow
print(f"Całkowita wartość zamówienia, to: {zamownienie}")
```

Zadanie 9

```
import random #biblioteka do liczb pseudolosowych
l1=random.randint(1,10) #losowanie liczby całkowitej z zakresu <1,10>
l2=random.randint(1,10)
wynik=l1*l2
liczba_u=input(f"Ile to jest {l1} * {l2} ?")
print(f"Odpowiedź użytkownika: {liczba_u}")
print(f"Odpowiedź komputera: {wynik}")
```

Zadanie 10

```
print("Potęgowanie liczb.")
pod=float(input("Podaj podstawę: "))
wyk=float(input("Podaj wykładnik: "))
wy=pod**wyk
print(f"Wynik wynosi: {wy}.")
```

Zadanie 11

```
tekst = input("Wprowadź tekst: ")
ile_znakow = len(tekst)          #len() sprawdza długość napisu
ile_spacji = tekst.count(" ")    #tekst.count(" ") zlicza liczbę wystąpień
spacji w napisie tekst
print(f"Liczba znaków: {ile_znakow}")
print(f"Liczba spacji: {ile_spacji}")
```

Zadanie 12

```
lista=[] #przykład bez pętli
print("Dodawanie do listy imion!")
lista.append(input("Podaj 1 imie: "))
lista.append(input("Podaj 2 imie: "))
lista.append(input("Podaj 3 imie: "))
lista.append(input("Podaj 4 imie: "))
lista.append(input("Podaj 5 imie:"))
print("Lista imion: ")
print(lista)
print("")
print(f"Cześć {lista[0]}")
print(f"Cześć {lista[1]}")
print(f"Cześć {lista[2]}")
print(f"Cześć {lista[3]}")
print(f"Cześć {lista[4]}")
```

Zadanie 13

```
#Program sumuje wszystkie liczby z obu list, wersja bez pętli.
lista1 = [1, 3, 2, 5]
lista2 = [4, 5, 1, 8]
suma =
lista1[0]+lista1[1]+lista1[2]+lista1[3]+lista2[0]+lista2[1]+lista2[2]+lista
2[3]
print(f"Suma liczb to {suma}")
```

Zadanie 14

```
#Wersja programu z pętlą
lista=[]
suma = 0
for i in range(1,8):
    lista.append(int(input(f"Podaj {i} liczbę: ")))
    suma = suma + i

print()
print("Elementy listy: ")
print(lista)
```

```
print()
srednia=suma/len(lista)
print(f"Suma liczb to {suma}")
print(f"Średnia liczb to {srednia}")

print()
print("Elementy listy w odwrotnej kolejności: ")
for i in range(6, -1, -1):
    print(lista[i], end=", ")
```

Zadanie 15

```
#Zbiory z założenia przechowują unikatowe dane
#Wersja bez pętli
zbior = set()
zbior.add(int(input("Podaj liczbę: ")))
zbior.add(int(input("Podaj liczbę: ")))
zbior.add(int(input("Podaj liczbę: ")))
zbior.add(int(input("Podaj liczbę: ")))
zbior.add(int(input("Podaj liczbę: ")))
ile = len(zbior) #ilość elementów zbioru = liczba unikatowych liczb

print(f"Unikatowych liczb jest {ile}")
```

Zadanie 16

```
zmienna1 = "jeden"
zmienna2 = "pięć"
zmienna3 = "siedem"
slownik={"jeden":1, "pięć":5, "siedem":7}

suma=slownik[zmienna1] + slownik[zmienna2] + slownik[zmienna3]
print(f"Suma liczb: {suma} ")
```

Zadanie 17

```
slownik = {"1":"jeden", "2":"dwa", "3":"trzy", "4":"cztery", "5":"pięć",
"6":"sześć", "7":"siedem", "8":"osiem", "9":"dziewięć"}
liczba = input("Podaj liczbe 4 cyforową :")
```

```
print(f"{slownik[liczba[0]]} {slownik[liczba[1]]} {slownik[liczba[2]]}
{slownik[liczba[3]]}")
```

Zadanie 18

```
import random # biblioteka do generowania liczb pseudolosowych

l1=random.randint(1,10) # losowanie liczby z zakresu <1, 10>
l2=random.randint(1,10)
wynik=l1*l2
liczba_g=int(input(f"Ile to jest {l1} * {l2} ?"))

print(f"Odpowiedź użytkownika: {liczba_g}")
print(f"Odpowiedź komputera: {wynik}")

if wynik == liczba_g:
    print("Użytkowniku podałeś prawidłowy wynik!")
else:
    print("Użytkowniku podałeś błędny wynik!")
```

Zadanie 19

```
liczba = int(input("Wprowadź liczbę: "))
if liczba % 2 == 0:
    print("Wprowadzona liczba jest parzysta!")
else:
    print("Wprowadzona liczba jest nieparzysta!")
```

Zadanie 20

```
# wersja 2 zmienne
zm1 = 5.0
zm2 = 2.0
print (f"Wartość zm1 = {zm1}, a zm2 = {zm2}")
if zm1 > zm2:
    print(f"Zmienna pierwsza ma większą wartość: {zm1}!")
else:
    print(f"Zmienna druga ma większą wartość: {zm2}!")

# wersja 3 zmienne (założenie zmienne są różne!)
z1 = 2.0
```

```
z2 = 3.0
z3 = 10.0
print (f"Wartość z1 = {z1}, z2 = {z2}, z3 = {z3}")
if z1 > z2:
    if z1 > z3:
        print("Zmienna z1 największa!")
    else:
        print("Zmienna z3 największa!")
elif z2 > z3:
    print("Zmienna z2 jest największa!")
else:
    print("Zmienna z3 jest największa!")
```

Zadanie 21

```
#wersja bez wbudwanych funkcji
a = 10
b = 2
c = 9

if a > b and a >c:
    print(a, end=" ")
    if b > c:
        print(b, c)
    else:
        print(c, b)
elif b > c and b > c:
    print(b, end=" ")
    if a > c:
        print(a, c)
    else:
        print(c, a)
elif c > a and c > b:
    print(c, end=" ")
    if a > b:
        print(a, b)
    else:
        print(b, a)
```

Zadanie 22

```
waga = float(input("Podaj swoją wagę [kg]: "))
wzrost = float(input("Podaj swój wzrost [m]: "))

bmi = waga/(wzrost**2)
print(f"Twoje BMI: {bmi}")
print()

if bmi >= 18.5 and bmi <=24.9:
    print("Waga prawidłowa!")
elif bmi <18.5:
    print("Niedowaga!")
else:
    print("Nadwaga!")
```

Zadanie 23

```
print("Program wyświetla liczby podzielne przez 6 z zakresu od <1,100>. ")
for i in range (1,101):
    if i%6 == 0:
       print(i, end=", ")
```

Zadanie 24

```
suma = 0
for i in range(1, 11):
    liczba = int(input(f"Podaj {i} liczbę: "))
    if liczba % 2 == 1:
       suma = suma + liczba

print(f"Suma liczb nieparzystych, to: {suma}")
```

Zadanie 25

```
imiona = []
for i in range (8):
    imiona.append(input("Podaj imie: "))

print()

for i in imiona:
    print(f"Witaj {i}.")
```

Zadanie 26

```
podstawa = int(input("Podaj podstawę: "))
wykladnik = int(input("Podaj wykładnik: "))

potega = 1

if wykladnik >= 0:
    for i in range(wykladnik):
        potega=potega*podstawa
    print(f"{podstawa}^{wykladnik}={potega}")
else:
    print("Nieprawidłowa wartość wykładnika!")
```

Zadanie 27

```
liczba = int(input("Podaj z jakiej liczby chcesz obliczyć silnie: "))
silnia = 1
if liczba >= 0:
    for i in range(1,liczba+1):
        silnia=silnia*i
    print(f"{liczba}! = {silnia}")
else:
    print("Wyznaczenie silni dotyczy liczb dodatnich oraz zera.")
```

Zadanie 28

```
import random
los = random.randint(1,100)

while True:
    liczba = int(input("Podaj wylosowaną liczbę: "))

    if liczba > los:
        print("Podałeś za dużą liczbę.")
    elif liczba < los:
        print("Podałeś za małą liczbę.")
    elif liczba == los:
        print("Gratulacje!")
        break
```

Literatura

Ceder N., *Python. Szybko i prosto*, Helion, Gliwice 2019.

Dawson M., *Python dla każdego. Podstawy programowania*, Helion, Gliwice 2014.

Gaddis T., *Python dla zupełnie początkujących*, Helion, Gliwice 2019.

Jaworski M., Lipiński S., *Python. Kurs programowania na prostych przykładach*, Helion, Gliwice 2018.

Jaworski M., *Python. Ćwiczenia praktyczne*, Helion, Gliwice 2019.

Lech P., *Python. Podstawy programowania*, WNT, Warszawa 2015.

Lutz M., *Python. Wprowadzenie*, Helion, Gliwice 2020.

Matthes E., *Python. Instrukcje dla programisty*, Helion, Gliwice 2023.

Miles R. *Python. Zacznij programować!,* Helion, Gliwice 2018.

Sarbicki G., *Python. Kurs dla nauczycieli i studentów*, Helion, Gliwice 2020.

Zajączkowski S., *Python. Ćwiczenia praktyczne*, Helion, Gliwice 2017.