

Python x 6



1. Wyrażenia
2. Zmienne i instrukcja przypisania
3. Formatowanie tekstu
4. Instrukcje warunkowe
5. Instrukcje iteracyjne, listy i tablice, pliki tekstowe
6. Funkcje

Po co Python?

Python jest jednym z języków programowania. Służy przekazaniu do komputera instrukcji opisujących co procesor ma wykonać. Instrukcje te mają postać tekstu i są nazywane kodem. Stąd pisanie programu nazywamy kodowaniem. Kod wpisujemy zwykle do edytora, specjalnego programu zwanego IDE (Integrated Development Environment) który zamieni nasz kod na sekwencję poleceń dla procesora, spowoduje wykonanie poleceń i ewentualnie wyświetli wyniki. Istnieją różne IDE dla Pythona, ale jeśli chcemy najszybsze i najprostsze rozwiązanie, to można wykorzystać proste IDE dostępne on-line i oczywiście bezpłatne. Ma ono sporo wad, tworzone programy trzeba skopiować i zapisać np. w notatniku, dla późniejszego wykorzystania (wklejenia), nie można też czytać i zapisywać plików

<https://www.programiz.com/python-programming/online-compiler/>

Python i wszystkie jego przybudówki zwane bibliotekami są projektem „open-source” czyli całkowicie bezpłatnym. Jest to język wysokiego poziomu, co oznacza, że stara się przypominać język naturalny. Python powstał pod koniec lat 1980-tych. Jego twórcą jest Guido van Rossum. Nazwa języka pochodzi od ulubionego przez Guido telewizyjnego serialu komediowego Monty Python. Język został udostępniony w 1991 roku i od tego czasu rozwija się pod względem ilości dodatkowych modułów i pakietów, które rozszerzają jego możliwości.

Wśród niektórych panuje przekonanie, że Python służy wyłącznie do pisania krótkich skryptów (krótkich programów) łączących "prawdziwy" kod napisany w takim języku jak C++. Jednak Python okazał się przydatny w realizacji najróżniejszych zadań. Python jest powszechnie wykorzystywany w takich firmach jak Google, NASA czy Netflix. Jest także jednym z podstawowych języków używanych w inżynierii danych, uczeniu maszynowym i sztucznej inteligencji.

Python jest powszechnie wykorzystywany w pracy naukowej w różnych dziedzinach. Przede wszystkim dlatego, że jest prosty w zapisie i czytaniu kodu. Jednocześnie im więcej osób go stosuje tym więcej powstaje specjalistycznych rozwiązań w formie bibliotek. Rośnie też grono użytkowników, dzięki którym można znaleźć szybko rozwiązanie dowolnego problemu. Rozwojowi języka służą także dwa rozwiązania, definiujące zupełnie inny sposób pracy przy tworzeniu programowania i analizie danych – *GitHub* i *Jupyter notebook*.

Z powyższych powodów Python stał się ulubionym językiem do automatyzacji zadań w złożonych systemach takich jak na przykład GIS. Jest także wykładany jako pierwszy język na wielu kierunkach studiów w tym przede wszystkim na kierunkach nieinformatycznych., a także uczony w szkołach.

Zanim zaczniemy pracę z Pythonem, sprawdzimy z jaką jego wersją pracujemy. W tym celu napiszemy i wykonamy prosty program,

```
1 # W jakiej wersji Pythona pracujemy
2 from platform import python_version
3
4 print(python_version())
5 print("Hello world")
```

```
3.6.10
Hello world
```

Program składa się z linii (w naszym programie jest ich 5), które będą kolejno wykonywane. Linia pierwsza nie będzie wykonywana bo jako poprzedzona znakiem `#` stanowi komentarz do naszego programu. Każda linia może być chwilowo wyłączona z działania programu przez jej poprzedzenie za pomocą `#`. Linia druga pokazuje w jaki sposób wykorzystuje się rozszerzenia Pythona. W tym przypadku z modułu `platform` (wspomniane wcześniej biblioteki zawierają zwykle wiele modułów) pobieramy funkcję `python_version`. Funkcja to taki mini program pythonowy, który możemy wywołać za pomocą jej nazwy. Funkcja albo coś wykonuje, albo zwraca jakąś wartość, w tym przypadku zwraca tekst zawierający aktualną wersję Pythona. Linia trzecia jest pusta. Często wstawiamy takie linie do programu aby zwiększyć jego czytelność. Linia czwarta zawiera jedną z najczęściej stosowanych funkcji pythona `print`. Tej funkcji nie trzeba importować bo jest ona wbudowana w podstawowy Python. Funkcja wyświetla na ekranie wartość, wartość zwracaną przez funkcję, tekst lub wartość wyrażenia. W linii czwartej wyświetlony jest tekst zwracany przez funkcję, w linii piątej tekst „Hello world”. Proszę zwrócić uwagę, że tekst zawiera apostrofy.

1. Wyrażenia

Jeżeli wykonamy w Pythonie operację,

```
1 print(5 - 3)
```

```
2
```

to Python zadziałał jak kalkulator. Nasze działanie obliczyło wynik, a funkcja `print` wyświetliła go na ekranie. Nasze działanie nosi nazwę **wyrażenia**, składa się z dwóch wartości (5 i 3) oraz znaku minus, które nazywamy **operatorem**. To co jest najważniejsze w wyrażeniu to, że jego wynikiem jest zawsze jedna wartość. Wyrażenia możemy tworzyć intuicyjnie tak jak poniżej,

```
1 print((50 - 5*6)/4+11 - 3)
```

```
13.0
```

Wynik jest oczywisty, ale proszę zauważyć, że choć wynik jest 13 wyświetlony został jako 13.0. Wynika to z tego, że wartości, którymi będziemy operować mogą być różnego typu. Trzy podstawowe typy to: liczby całkowite (np. 6, -2, 1234), liczby rzeczywiste (np. 13.0, -123.34) i tekst, który będziemy nazywali **string** np. („To jest to”, „Liczba wynosi 2.34”). Proszę zauważyć, że w przypadku stringów

możemy używać pojedyncze lub podwójne apostrofy. Typ całkowity nazywa się **integer**, a rzeczywisty **float**.

Podstawowe typy danych w Pythonie

Typ danych	Przykład
Całkowite (<u>integer</u>)	-12, 0, 3, 234, 17
Rzeczywiste (<u>float</u>)	-1.30, 0.0, 12.01, 13.8
Tekstowe (<u>string</u>)	'klasa', 'A', 'koniec programu nr. 3',

Typ danych możemy zawsze sprawdzić za pomocą wbudowanej funkcji **type()**

```
1 print(type(5 - 3))
```

```
<class 'int'>
```

```
1 print(type((50 - 5*6)/4+11 - 3))
```

```
<class 'float'>
```

```
1 print(type('To jest wynik:'))
```

```
<class 'str'>
```

Zarówno funkcja `print()` i `type()` ma nawiasy, w których umieszczamy jej argumenty. Funkcja `print` może nie zawierać argumentu. Linia będzie pusta i nastąpi przejście do następnej linii. Wyświetlone skróty typów `int`, `float`, `str` odpowiadają funkcjom **int()**, **float()** i **str()**, za pomocą których możemy zmieniać typ wartości,

```
1 print(float(5 - 3))
```

```
2.0
```

```
1 print(int((50 - 5*6)/4+11 - 3))
```

```
13
```

Operatory, które wykorzystujemy do tworzenia wyrażeń algebraicznych w większości mają standardowy charakter. Istnieje jednak parę wyjątków i kilka operatorów dodatkowych.

Operatory algebraiczne w Pythonie

Operator	Operacja	Przykład	Wynik
**	potęgowanie	2 ** 3	8
%	reszta z dzielenia	22 % 8	6
//	dzielenie całkowite	22 // 8	2
/	dzielenie	22 / 8	2.75
*	mnożenie	3 * 4	12
-	odejmowanie	7 - 5	2
+	dodawanie	2 + 3	5

Nie wszystkie funkcje są wbudowane i trzeba je zaimportować. Przykładem takich funkcji i stałych są funkcje i stałe matematyczne modułu **math**. Import modułu odbywa się za pomocą linii:

```
1 from math import *
```

Jest to jeden ze sposobów, który daje nam dostęp do wszystkich funkcji i stałych tego modułu.

Jeśli chcielibyśmy policzyć powierzchnię w km² koła o promieniu 500m to wyrażenie miałoby postać.

```
1 print(pi*500**2/1000000)
```

0.7853981633974483

Zadanie jest trywialne, ale proszę zwrócić uwagę na parę szczegółów. Po pierwsze wykorzystaliśmy stałą **pi**. Po drugie nie musieliśmy stosować nawiasów, bo potęgowanie jest wykonywane jako pierwsze. Po trzecie używanie liczb z wieloma zerami może powodować pomyłki lepiej jest używać tzw. notacji naukowej w której liczby bardzo duże lub małe zapisujemy jako (e może być pisane małą lub dużą literą),

liczba	notacja naukowa
0,1	1E-1
1	1E0
10	1E1
123	1,23E2
98765	9,8765E4
0,045	4,5E-2
0,00045	4,5E-4

Na koniec wartość wynikową warto by było zaokrąglić, użyjemy funkcji wbudowanej **round**, która posiada dwa parametry wartość i liczbę zer po kropce. Ostatecznie nasze wyrażenie przyjmie postać,

```
1 print(round(pi*500**2*1e-6,3))
```

0.785

Przykład 1

Oblicz gęstość zaludnienia (liczba mieszkańców / 1 km²) woj. pomorskiego o powierzchni 18310 km² jeśli liczba jej mieszkańców wynosi 2.305 miliona.

Rozwiązanie

Wyrażenia można tworzyć także za pomocą operatorów porównania i logicznych. Przedstawione zostały one w tabelce poniżej. Operatory logiczne **and** i **or** zachowują się zgodnie z zasadami logiki. Należy zwrócić uwagę, że operator równości ma dwa znaki **==**.

```
1 print(7>2)
```

True

Wyrażenia porównania i logiczne przyjmują wartość prawdy i fałszu (**True**, **False**) i mają typ **bool**.

```
1 print(type(7>2))
```

<class 'bool'>

Można je łączyć z operatorami algebraicznymi, wtedy w wyrażeniu False jest zastępowane przez 0, a True przez 1.

```
1 print((7>2)+4)
```

5

Analogicznie zastępowanie może być w drugą stronę, liczba, która znajdzie się w miejscu typu bool, zostanie zinterpretowana jako False gdy jest równa 0 i jako True gdy nie jest równa 0. Ma sens zatem wyrażenie.

```
1 print(bool(0 and 22))
```

False

Proszę zwrócić uwagę, że istnieje funkcja **bool()** do konwersji do typu boolowskiego. Wynikiem operacji 0 and 22 będzie oczywiście zero,

```
1 print(0 and 22)
```

0

Tu 0 jest interpretowane jako False a 22 jako True.

Operatory porównania i logiczne w Pythonie

Operator	Znaczenie / wynik
==	równa się
!=	różne
<	mniej
>	więcej
<=	mniej lub równe
>=	więcej lub równe
True and True	True
True and False	False
False and True	False
False and False	False
True or True	True
True or False	True
False or True	True
False or False	False

Wspomniane wcześniej funkcje modułu **math** zawierają większość funkcji matematycznych, potrzebnych do typowych obliczeń. Najczęściej wykorzystywane zostały zestawione w tabeli poniżej. Wśród nich znajdują się funkcje trygonometryczne. Należy zapamiętać, że argumentami tych funkcji nie są stopnie, a radiany. Aby się nimi posługiwać należy pamiętać, że **2 * pi * radianów = 360 stopni**. Moduł matematyczny zawiera funkcje do zamiany stopni na radiany i radianów na stopnie.

```
1 from math import *
```

```
1 stopnie1=30
2 radiany1=radians(stopnie1)
3 print(stopnie1,round(radiany1,3),round(sin(radiany1),3))
```

30 0.524 0.5

Proszę zwrócić uwagę na umieszczanie jednej funkcji w drugiej i wykorzystanie funkcji round. Zamiana radianów na stopnie odbywa się w następujący sposób.

```
1 stopnie2=degrees(radiany1)
2 print(stopnie2)
```

29.999999999999996

Wynik inny od 30 jest spowodowany zapisem dwójkowym w komputerze.

Podstawowe funkcje modułu math

Funkcje modułu math	Opis funkcji
<code>ceil(x)</code>	najmniejsza liczba całkowita nie mniejsza od x
<code>floor(x)</code>	największa liczba całkowita nie mniejsza od x
<code>exp(x)</code>	e do potęgi x
<code>log(x)</code>	logarytm naturalny z x
<code>log10(x)</code>	logarytm dziesiętny z x
<code>pow(x,y)</code>	wartość $x^{**}y$
<code>sqrt(x)</code>	pierwiastek kwadratowy z x
<code>pi</code>	stała 3.141599265359
<code>e</code>	stała 2.718281882846
<code>sin(x)</code>	sinus x (x w radianach)
<code>cos(x)</code>	cosinus x (x w radianach)
<code>tan(x)</code>	tangens z (w radianach)
<code>acos(y)</code>	zwraca arcus cosinus (w radianach)
<code>atan(y)</code>	zwraca arcus tangens (w radianach)
<code>degrees(x)</code>	zamienia radiany na stopnie
<code>radians(x)</code>	zamienia stopnie na radiany

Przykład 2

Nachylenie zbocza można obliczyć za pomocą wyrażenia,

$S[\text{kąty}] = \arctan(\text{zmiana wysokości na jednostkę długości})$

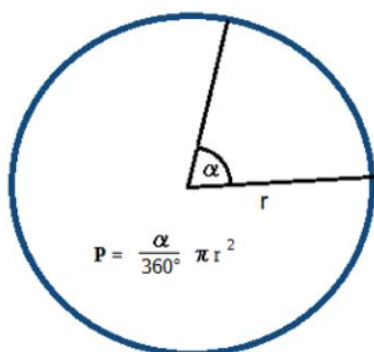
Jakie jest nachylenie w stopniach jeśli różnica wysokości wynosi 100 m przy dystansie 1 km.

Rozwiązanie

Przykład 3

Wiedząc, że pole wycinka koła można obliczyć ze wzoru poniżej. Oblicz pole wycinka, jeśli

$\alpha = 0.89$ rad, a $r = 3$.



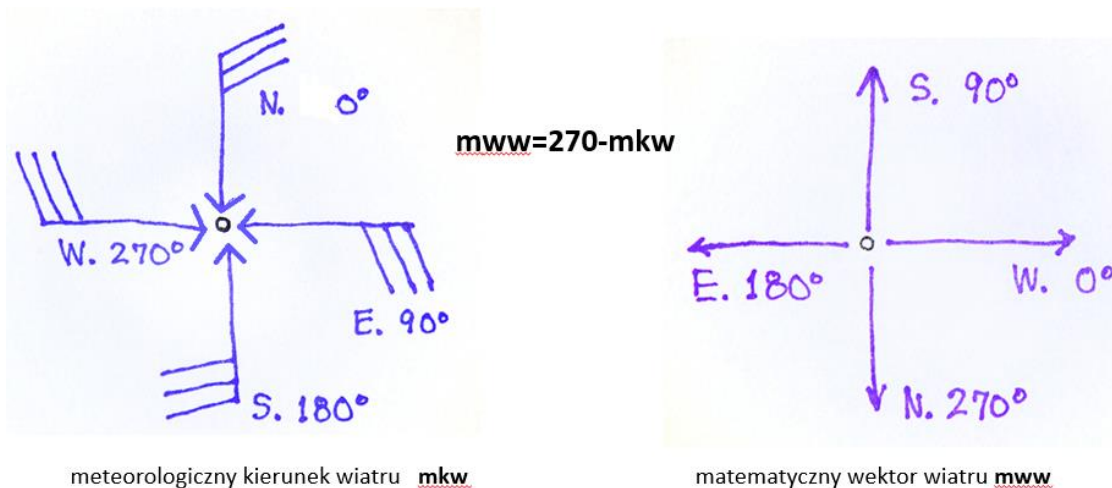
Rozwiązanie

Przykład 4

Oblicz składowe wektora wiatru, jeżeli jego kierunek (meteorologiczny) wynosi 50° a prędkość 6 m/s. Kierunek meteorologiczny, jest to kierunek z którego wiatr wieje. Aby obliczyć składowe działania wiatru, należy przeliczyć kierunek na kierunek matematycznego wektora wiatru, za pomocą wyrażenia przedstawionego na rysunku. Jeśli wynik jest ujemny należy dodać 360. Wyrażenia na składowe będzie miało postać:

$$u = \text{prędkość wiatru} * \cos(\text{mww})$$

$$v = \text{prędkość wiatru} * \sin(\text{mww})$$



Rozwiązanie

Oprócz modułu **math** poznamy jeszcze moduł do tworzenia liczb losowych, a właściwie pseudolosowych. Aby z niego korzystać musimy go uruchomić za pomocą wyrażenia:

```
1 from random import *
```

W module tym istnieje szereg funkcji zawierających różne funkcje losowe, oraz funkcja powodująca, że przy kolejnych uruchamianiach programu będziemy otrzymywać tę samą sekwencję liczb losowych. Podstawowe funkcje zestawione są w tabeli poniżej.

Funkcja modułu <u>random</u>	Opis funkcji
<u>randint(a,b)</u>	losowa liczba całkowita od a do b
<u>random()</u>	losowa liczba rzeczywista (0-1)
<u>gauss(a,b)</u>	losowa liczba z rozkładu normalnego o średniej a i odchyleniu standardowym b
<u>uniform(a,b)</u>	losowa liczba rzeczywista od a do b
<u>seed(a)</u>	tworzenie powtarzalnej serii liczb losowych

Funkcje działają w następujący sposób:

```

1 # Liczba całkowita od 1 do 10
2 print(randint(1,10),end=', ');print(randint(1,10),end=', ')
3 print(randint(1,10),end=', ');print(randint(1,10),end=', ')
4 print(randint(1,10),end=', ');print(randint(1,10),end=', ')
5 print(randint(1,10),end=', ');print(randint(1,10),end=', ')
6 print(randint(1,10),end=', ');print(randint(1,10))

```

6,1,8,7,6,8,8,1,10,8

W programie tym 10 razy zastosowano funkcję print. Zamiast pisać je w oddzielnych liniach, zapisano po dwie w jednej rozdzielając je średnikiem. Wykonanie funkcji print domyślnie powoduje przejście do nowej linii, można to zmienić za pomocą argumentu end= i wpisując dowolny tekst. Takie dwa zapisy są równoważne i pokazują jak działa funkcja random i średnik:

```

1 # sposób 1
2 print(random(),random())
3 print(random(),random())
4 # sposób 2
5 print()
6 print(random(),random());print(random(),random())

```

0.24893488828640298 0.16266914300282997
0.2187983775104979 0.26272211234411835

0.5203472275275995 0.32939484187575596
0.9979950477695798 0.32823099614121143

Funkcja seed działa w następujący sposób:

```

1 seed(120)
2 print(uniform(0,100))
3 print(uniform(0,100))

```

51.493795400774914
18.86481713153758

```

1 seed(120)
2 print(uniform(0,100))
3 print(uniform(0,100))

```

51.493795400774914
18.86481713153758

Każde użycie seed inicjuje nową sekwencję liczb pseudolosowych, wartość liczbową (120) określa wartość początku sekwencji.

Wyrażenia mogą być też budowane z tekstów, wykorzystując operację ich dodawania. Dodanie jednego tekstu do drugiego powoduje ich połączenie. Operator + jest operatorem, który w zależności od kontekstu działa w inny sposób.

```

1 print('Wylosowana liczba to '+str(randint(1,49))+' oraz '
2      +str(randint(1,49)))

```

Wylosowana liczba to 11 oraz 39

Połączono tutaj cztery teksty przy czym dwa powstały w rezultacie zamiany liczby całkowitej na tekst za pomocą funkcji str().

Istnieje istotna różnica pomiędzy wartościami typu int, float, bool i tekstami (typ str). Te pierwsze zawierają jedną wartość i są niepodzielne. W odróżnieniu od nich tekst posiada strukturę w postaci sekwencji znaków.

```
1 print('To jest tekst')
2 print(type('To jest tekst'))
```

```
To jest tekst
<class 'str'>
```

Istnieje funkcja **len()**, która zwraca liczbę znaków w tekście,

```
1 print(len('To jest tekst'))
```

```
13
```

Dostęp do każdego elementu struktury tekstu jest możliwy dzięki **indeksowaniu**, które przyjmuje postać,

TEKST[indeks]

indeksy zaczynają się zawsze od zera, czyli aby wydrukować czwartą literę należy użyć indeksu 3 (pierwsza 0, druga 1, trzecia 2, czwarta 3). Indeksy ujemne wskazują znaki od końca.

```
1 print('To jest tekst'[3])
```

```
j
```

```
1 print('To jest tekst'[-3])
```

```
k
```

Istnieje także możliwość otrzymania dowolnej sekwencji znaków tekstu za pomocą ich wycinania, operacja ta nosi nazwę **slicing**. Przyjmuje ona postać,

TEKST[indeks1:indeks2]

Zasada jest taka, że **indeks1** jest pierwszym znakiem wyniku operacji a **indeks2-1** ostatnim. Wycinek zawiera **indeks2 - indeks1** znaków (jeśli oba indeksy są dodatnie). Możliwe są różne warianty stosowania tej metody,

```
1 print('[1:6]', 'To jest tekst'[1:6])
2 print('[ :6]', 'To jest tekst'[:6])
3 print('[1:]', 'To jest tekst'[1:])
4 print('[ :]', 'To jest tekst'[:])
5 print('[5:-2]', 'To jest tekst'[5:-2])
```

```
[1:6] o jes
[ :6] To jes
[1:] o jest tekst
[ :] To jest tekst
[5:-2] st tek
```

2. Zmienne i instrukcja przypisania

Wartości, które mogą być różnego typu (na razie poznaliśmy typy int, float, bool i str), są obiektami, podstawowymi klockami z których zbudowany jest Python. **Zmienna** to nazwa **przypisana** do tych obiektów. Zmienna powstaje w procesie przypisania,

`nazwa_zmiennej = wartość`

Instrukcja powiązania nazwy z wartością to **instrukcja przypisania** (znak `=`). Nazwa zmiennej jest zawsze z lewej strony, a wartość będąca najczęściej rezultatem wyrażenia z prawej strony instrukcji przypisania. Zmienna pozwala tworzyć uniwersalne wyrażenia, w których zmienne reprezentują dowolne wartości. W przykładzie poniżej tworzymy uniwersalny program, który pozwala na obliczenie odległości pomiędzy dwoma punktami.

```
1 x1, y1= 123.4, 13.2
2 x2, y2= 17.1, 345.2
3
4 dyst=sqrt((x2-x1)**2+(y2-y1)**2)
5 print(round(dyst,1))
```

348.6

Program pokazuje jak wykorzystujemy zmienną do wprowadzenia wartości danych (linia 1,2). Kolejny krok to obliczenie wartości wyrażenia i przypisania go zmiennej, która może być wykorzystana w kolejnych liniach programu. Jeśli chcemy obliczyć dystans dla innych współrzędnych, wystarczy podać inne wartości w pierwszych dwóch liniach. Z lewej strony instrukcji przypisania mamy zmienne (x1,y1,x2,y2,dyst), a z prawej wartości, lub wyrażenia, które zwracają wartości (linia 4).

Istnieją pewne zasady tworzenia nazw zmiennych. Zaczynają się one zawsze literą (bez polskich), a po nich mogą następować litery (bez polskich), cyfry oraz podkreślnik `_`. Nie można w nazwach używać takich znaków jak: `#`, `&`, `@` i `%`. Python jest językiem „case sensitive” oznacza to, że Temperatura i temperatura to dwie różne nazwy. Ponadto Python zawiera pewną liczbę zarezerwowanych słów (keyword), które nie mogą pełnić roli nazw. Najważniejsze to: `and`, `as`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `for`, `from`, `global`, `if`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `return`, `try`, `with`, `while`.

Przeanalizujemy poniższy program,

```
1 x1=3
2 print(x1,type(x1))
3 x2=8.3
4 print(x2,type(x2))
5 x1=x1/x2
6 print(x1,type(x1))
```

```
3 <class 'int'>
8.3 <class 'float'>
0.36144578313253006 <class 'float'>
```

W programie tym mamy dwie zmienne x1 i x2. W linii (1) nastąpiło przypisanie x1 wartości 3. Ponieważ wartość była całkowita typ x1 jest int. W linii (3) przypisano wartość 8.3 do zmiennej x2.

Przykład 5

Oblicz dystans pomiędzy Lizboną (38°42'0"N, 9°11'0"W) a Gdańskiem (54°20'51"N, 18°38'43"E). Zastosuj wyrażenie:

$$D = \arccos(\sin \varphi_1 \cdot \sin \varphi_2 + \cos \varphi_1 \cdot \cos \varphi_2 \cdot \cos \Delta\lambda) \cdot R$$

gdzie: φ_1, φ_2 - szerokości geog., $\Delta\lambda$ - różnica dług. geog., R - promień Ziemi (średni = 6371 km)

Rozwiązanie

Przykład 6

Miarą opisującą kształt linii jest jej krętość (sinusoity). Oblicza się ją za pomocą wyrażenia:

$$SIN = D_L / D_{SK}$$

gdzie: D_L długość linii (suma wszystkich jej odcinków), a D_{SK} to odległość pomiędzy początkiem, a końcem linii. Oblicz krętość linii o długości 1010 m jeśli jej początek ma współrzędne (x=429061m, y=747738m), a koniec (x=428523m, y=747822m). Odległość oblicz z wyrażenia,

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

gdzie: x_1, y_1 i x_2, y_2 współrzędne punktów.

Rozwiązanie

Wprowadzanie danych może odbywać się interaktywnie, przez wpisanie wartości w ramce za pomocą funkcji,

zmienna = input('Tekst nad ramką ')

Wykonanie tej funkcji zawiesza program i wyświetla ramkę, w którą wpisujemy wartość i naciskamy ENTER, po czym program kontynuuje działanie,

```
1 xx = input('Podaj wartość xx:')
```

Podaj wartość xx: 3.4

Ze względu na to, że funkcja **input** zwraca tekst (string) jej rezultat musi być zmieniony w liczbę, jeśli istnieje taka potrzeba,

```
1 xx = input('Podaj wartość xx:')
2 print(xx,type(xx))
3 xx = float(input('Podaj wartość xx:'))
4 print(xx,type(xx))
```

Podaj wartość xx:3.4

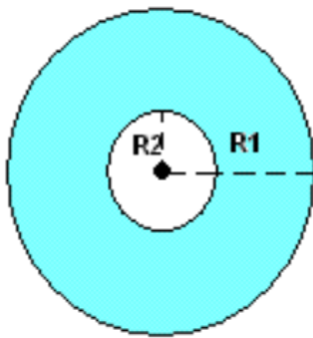
3.4 <class 'str'>

Podaj wartość xx:2.2

2.2 <class 'float'>

Przykład 7

Oblicz powierzchnię pierścienia na podstawie poniższego wzoru. Użyj dwóch wprowadzanych zmiennych r1 i r2 do przechowania wartości dwóch promieni.



$$\text{Area} = \pi (R1^2 - R2^2)$$

Rozwiązanie

Przykład 8

Oblicz prędkość samochodu po 30 sekundach od startu, jeśli jego przyspieszenie wynosi 1.5 m s^{-2} . Zapisz czas w zmiennej t, by można go było swobodnie modyfikować.

$$v = v0 + at, \text{ gdzie } a = \text{const}$$

Rozwiązanie

3. Formatowanie tekstu

Formatowanie pozwala na elastyczne określenie postaci tekstu (stringu) w naszym programie. W Pythonie istnieją obecnie trzy różne sposoby formatowania. Zapoznamy się z najnowszą metodą obecną od wersji Pythona 6.3 zwaną **f-string**. W porównaniu z innymi metodami charakteryzuje ją duża prostota i szybkość działania.

Każde wyrażenie **f-string** składa się z dwóch części, pierwszą jest znak **f** lub **F**, drugą tekst w pojedynczych, podwójnych lub potrójnych apostrofach. W tekście umieszcza się **{ }** do wstawiania zmiennych, które mogą być formatowane (po znaku **:** umieszczonym za zmienną). Wygląda to w taki sposób,

```
1 x=12
2
3 print(f"to jest tekst - {x} koniec")
```

to jest tekst - 12 koniec

Każdy element tekstu (znak) zostaje wprowadzony wraz z wartościami zmiennej w nawiasach. Wartości zmiennej zostają wprowadzone w miejscu nawiasów. W nawiasach mogą się także znaleźć wyrażenia.

```
1 x=12;y=234
2
3 ss= f'wynik = {x*y}'
4 print(ss)
```

wynik = 2808

Za pomocą specjalnego znaku **\n** można wymusić przejście tekstu do następnego wiersza, znak specjalny nie jest zapisywany.

```
1 x=12;y=234
2
3 ss=f'wynik1 = {x}\nwynik2 = {y}'
4
5 print(ss)
```

wynik1 = 12
wynik2 = 234

Zmienna może być oczywiście tekstem, ale jeśli chcemy aby zawierała ona w sformatowanym tekście cudzysłowy nie musimy wprowadzać ich oddzielnie, ale zastosować zapis,

```

1 x='P20'
2
3 s=f'to jest punkt : {x!r}'
4 print(s)

```

to jest punkt : 'P20'

Liczby całkowite mogą być formatowane za pomocą podania liczby pól w których zostaną wpisane. Liczbę pól podajemy po dwukropku umieszczonym za zmienną. Domyślnie będą justowane do prawej strony zaplanowanej przestrzeni, można to zmienić za pomocą znaku < po dwukropku. Możliwe jest także wprowadzenie zer wiodących.

```

1 val = 25
2 print(' 12345678')
3 print(f'0|{val}|')
4 print(f'1|{val:8}|')
5 print(f'2|{val:4d}|')
6 print(f'3|{val:<8d}|')
7 print(f'4|{val:<4d}|')
8 print(f'5|{val:08d}|')
9 print(f'6|{val:04d}|')

```

```

12345678
0|25|
1|      25|
2|  25|
3|25      |
4|25  |
5|00000025|
6|0025|

```

Liczby w górnym wierszu oznaczają pola, czyli miejsca na znak. W wierszu na wydruku o numerze 0, w nawiasach znajduje się tylko zmienna. Zostanie ona wstawiona w sformatowany tekst i zajmie tyle pól ile ma znaków. Pionowa kreska, która jest kontynuacją tekstu zostanie zapisana zaraz za liczbą. W wierszu numer 1 mamy zarezerwowane 8 pól, litera **d** oznacza liczbę całkowitą (mogłoby jej nie być). Zajętych zostanie 8 pól, a liczba zostanie dosunięta do ich prawej krawędzi. W wierszu 3 i 4 użycie znaku < po dwukropku spowodowało dosunięcie do lewej krawędzi pól. Natomiast 5 i 6 wiersz wydruku pokazuje w jaki sposób wprowadzamy zera wiodące.

Liczby rzeczywiste są formatowane w podobny sposób. Różnica polega na uwzględnieniu liczby miejsc po kropce. Realizuje się to za pomocą formatu **8.2f**, gdzie 8 to liczba pól, a 2 to liczba znaków po kropce zaokrąglonych matematycznie. Może też być stosowany format **.2f**, który powoduje jedynie zaokrąglenie liczby, bez przydzielania pól. Symbol **f** może być zastąpiony symbolem **e**, który daje format naukowy np. 2.533e+01. Poniżej przedstawiono przykłady formatowania liczb rzeczywistych.

Istnieje szereg dodatkowych możliwości formatowania: znak **^** daje możliwość centrowania zmiennej w dostępnym polu, **+** powoduje, że wartości dodatnie będą poprzedzane znakiem plus.


```

1 val = 25.33333
2 print(' 12345678')
3 print(f'a|{val:e}|')
4 print(f'0|{val:.3f}|')
5 print(f'1|{val:8.3f}|')
6 print(f'2|{val:5.2f}|')
7 print(f'3|{val:<8.3f}|')
8 print(f'4|{val:<5.2f}|')
9 print(f'5|{val:08.3f}|')
10 print(f'6|{val:010.1e}|')

```

```

12345678
a|2.533333e+01|
0|25.333|
1| 25.333|
2|25.33|
3|25.333 |
4|25.33|
5|0025.333|
6|0002.5e+01|

```

Formatowanie tekstu odbywa się na tych samych zasadach, ale bez litery po dwukropku. Justowanie do prawego końca pól wymaga znaku `>`. Nie ma też możliwości wstawiania zer lub innych znaków wiodących.

```

1 val = 'pnk_25'
2 print(' 12345678')
3 print(f'0|{val}|')
4 print(f'1|{val:>8}|')
5 print(f'3|{val:<8}|')

```

```

12345678
0|pnk_25|
1|  pnk_25|
3|pnk_25 |

```

Przykład 9

Napisz program, który poprosi o wpisanie z klawiatury: imię (zmienna typu tekst), wiek (wartość całkowita) i wzrost (wartość rzeczywista) oraz wypisze te informacje w poniższy sposób:

Imię ma x lat oraz y metra wzrostu.

Przykład:

Adam ma 50 lat oraz 1.85 metra wzrostu.

Rozwiązanie

Przykład 10

Wylosuj sześć liczb rzeczywistych w zakresie 0 – 100. Utwórz pojedynczy tekst (string), który po zastosowaniu do niego funkcji **print** utworzy 3 ciągłe linie zaokrąglone do trzech miejsc po kropce. Powinien on wyglądać np. w taki sposób:

34.123,12.111

2.123,0.128

99.021,9.129

[Rozwiązanie](#)

Przykład 11

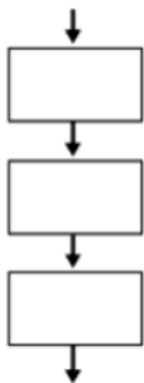
Utwórz w wydruku (pojedyncza funkcja print) choinkę w postaci:

```
  **
 ***
****
*****
*****
*****
```

[Rozwiązanie](#)

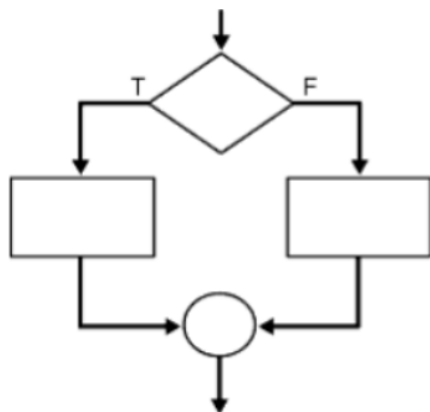
4. Instrukcje warunkowe

Wszystkie programy prezentowane do tej pory były wykonywane linia po linii. Ten sposób programowania, nazywa się „straight-line programing” i może on być przedstawiony jako sekwencja linii programu, które są wykonywane po kolei bez wyjątku,

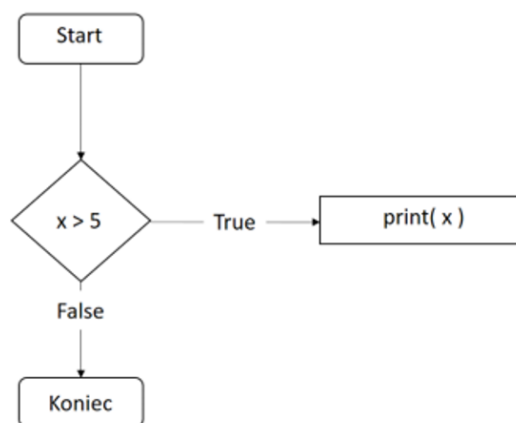


Kolejność wykonywania linii programu określa „flow control”. Można to sobie wyobrazić jako przepływ wody przez strukturę programu. Startujemy z górnej linii (nalewamy do niej wody),

następnie woda spływa w dół przez wszystkie linie. Wyobraźmy teraz sobie, że w pewnym miejscu, nasz przepływ ma możliwość popłynięcia różnymi kanałami. Zależnie od sytuacji program będzie realizował różne bloki linii kodu. Sytuację określać będzie warunek, którego wynikiem może być wartość TRUE albo FALSE. W zależności od wyniku program będzie realizowany w inny sposób. Poniżej przedstawiono schemat instrukcji warunkowej, po realizacji warunku program się rozgałęzia tworząc „branching program”. Instrukcja ta składa się zwykle z trzech elementów: testu w postaci wyrażenia zwracającego prawdę lub fałsz (na rysunku rąb), bloku kodu, który jest realizowany, jeśli wynikiem wyrażenia jest prawda (prostokąt z lewej strony) i bloku kodu, który jest realizowany, jeśli wynikiem wyrażenia jest fałsz (prostokąt z prawej strony). Koniec instrukcji iteracyjnej, to powrót do sekwencyjnego wykonywania linii programu (okrąg).



Najprostszą instrukcją warunkową jest **if**,



W tej instrukcji mamy test $x < 5$, jeśli wynikiem testu będzie True to wartość x będzie wydrukowana, jeśli False to instrukcja kończy pracę. Instrukcja będzie miała postać:

if (test):

kod do wykonania

Proszę zwrócić uwagę na następujące elementy: słowo kluczowe **if**, po którym znajduje się wyrażenie testu, linię kończy znak **:**. Jeśli wynikiem wyrażenia jest prawda, wtedy zostaje wykonany kod do wykonania, znajdujący się pod linią z testem. Ten kod może składać się z szeregu linii, a identyfikuje je

identyczne wcięcie (przesunięcie linii względem położenia linii z testem). Instrukcja będzie miała postać,

```
1 x=7
2 if x>5:
3     print(x)
```

7

W tej wersji wykonywany jest blok, jeśli warunek testu został spełniony. Drugą wersją instrukcji warunkowej jest jej rozgałęzienie na dwa bloki. Nosi ona nazwę **if ... else**, i ma postać

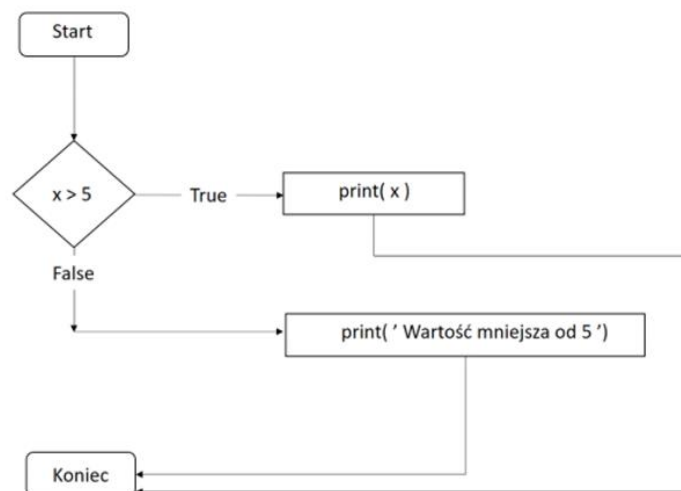
if (test):

kod do wykonania

else:

inny kod do wykonania

Jej schemat można zobrazować w następujący sposób,



Modyfikacja polega na tym, że jeżeli warunek nie jest spełniony wykonywany jest inny blok kodu.

```
1 x=4
2 if x>5:
3     print(x)
4 else:
5     print('Wartość mniejsza od 5')
```

Wartość mniejsza od 5

Blok kodu pod instrukcją warunkową, może zawierać inną instrukcję warunkową. Mówimy, że tak instrukcja jest zagnieżdżona. Przykładowo rozpatrzmy program klasyfikujący zmienną na trzy klasy. Klasę pierwszą przypisujemy wartościom mniejszym od 100, drugą wartościom większym lub równym 100 i mniejszym od 150, a trzecią pozostałym.

```

1 x=80
2 if x < 100:
3     print('Klasa 1')
4 else:
5     if x>= 100 and x<150:
6         print('klasa 2')
7     else:
8         print ('Klasa 3')

```

Klasa 1

Trzecią i najbardziej złożoną wersją instrukcji warunkowej jest **if ...elif... else** , ma ona postać,

if (test):

kod 1 do wykonania

elif (test):

kod 2 do wykonania

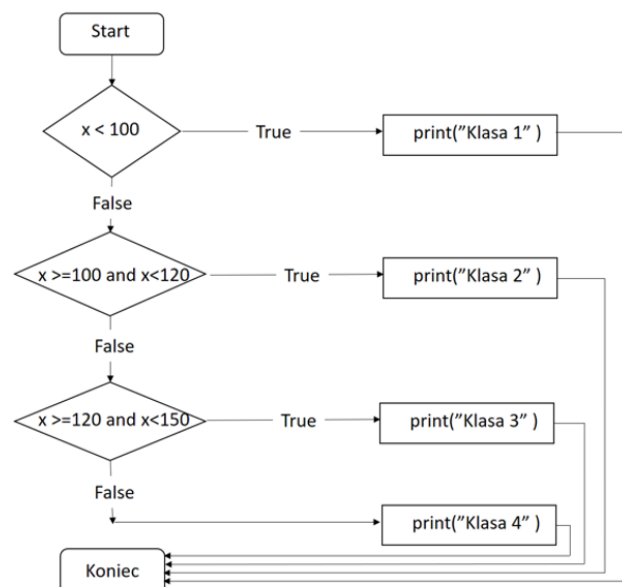
elif (test):

kod 3 do wykonania

else:

inny kod do wykonania

Po pierwszym teście z **if** jeśli wynik jest False, instrukcja wykonuje kolejne testy. Jeśli test jest True instrukcja wykonuje blok i kończy działanie. Schemat będzie miał postać,



a jego realizacja będzie wyglądać tak,

```

1 x=122
2 if x < 100:
3     print('Klasa 1')
4 elif x >= 100 and x < 120:
5     print('Klasa 2')
6 elif x >= 120 and x < 150:
7     print('Klasa 3')
8 else:
9     print('Klasa 4')

```

Klasa 3

Przykład 12

Napisz program, który po podaniu wieku psa (w latach) przelicza jego wiek na lata ludzkie według następujących zasad. Pierwszy rok psa to 15 lat ludzkich. Drugi rok psa to około 9 ludzkich lat. Każdy następny rok to 5 lat ludzkich.

[Rozwiązanie](#)

Przykład 13

Sprawdź czy z wylosowanych 3 długości odcinków (wartości całkowite od 1 do 100) można zbudować trójkąt. Podaj na wydruku informację, czy będzie on różnoramienny, równoramienny czy równoboczny. Zastosuj *Triangle Inequality Theorem* która określa warunek, że z 3 odcinków a,b,c można zbudować trójkąt,

- $a+b>c$
- $a+c>b$
- $b+c>a$

[Rozwiązanie](#)

Przykład 14

Napisz program, który służy do wyznaczenia powierzchni terrarium dla pytona w zależności od jego długości (podawana przez użytkownika). Minimalna powierzchnia wynosi 0.5 m^2 na 1 m pytona do 3 m, na każdy metr powyżej należy zapewnić 0.75 m^2 .

[Rozwiązanie](#)

5. Instrukcje iteracyjne, listy i tablice, pliki tekstowe

Założmy, że naszym zadaniem jest sumowanie liczb naturalnych 1,2,3 tak długo, aż ich suma jest mniejsza od 100. Zaproponowany program realizuje to w kolejnych liniach przypisując do poprzednio obliczonej sumy nową liczbę naturalną i drukując liczbę i sumę.

```
1 suma=0
2 suma=suma+1;print(1,suma); # suma=0+1=1
3 suma=suma+2;print(2,suma); # suma=1+2=3
4 suma=suma+3;print(3,suma); # suma=3+3=6
5 suma=suma+4;print(4,suma); # suma=6+4=10
6 suma=suma+5;print(5,suma); # suma=10+5=15
7 suma=suma+6;print(6,suma); # suma=15+6=21
8 suma=suma+7;print(7,suma); # suma=21+7=28
9 suma=suma+8;print(8,suma); # suma=28+8=36
10 suma=suma+9;print(9,suma); # suma=36+9=45
11 suma=suma+10;print(10,suma); # suma=45+10=55
12 suma=suma+11;print(11,suma); # suma=55+11=66
13 suma=suma+12;print(12,suma); # suma=66+12=78
14 suma=suma+13;print(13,suma); # suma=78+13=91
15 suma=suma+14;print(14,suma); # suma=91+14=105
```

Wynikiem programu jest wydruk,

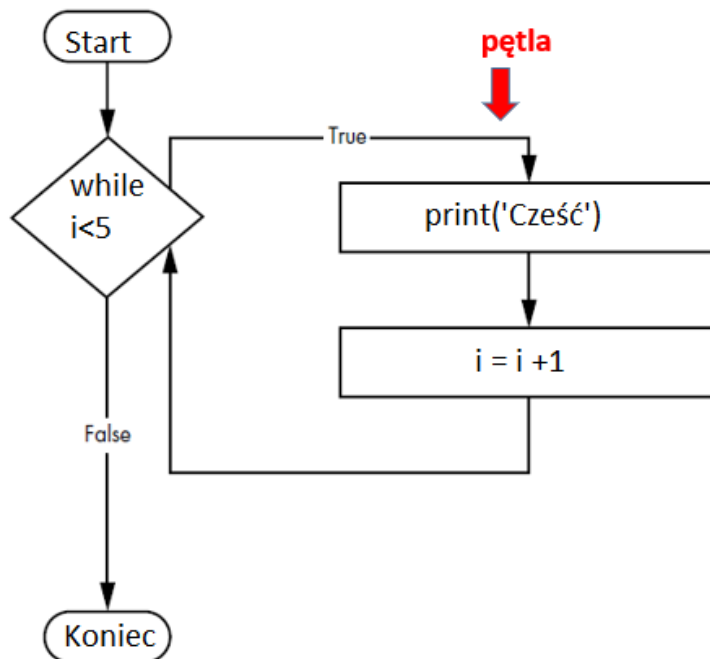
```
1 1
2 3
3 6
4 10
5 15
6 21
7 28
8 36
9 45
10 55
11 66
12 78
13 91
14 105
```

Aktualizowanie zmiennej (suma) przez przypisanie do niej jej starej wartości zmodyfikowanej do nowej, w tym przypadku przez dodanie kolejnej liczby naturalnej, jest często stosowanym mechanizmem. Tak napisany program, choć daje rozwiązanie, jest powtarzaniem tej samej konstrukcji. Można to zrobić prościej za pomocą instrukcji powodujących powtarzanie kodu czyli tworzenia tzw. pętli czy iteracji. W Pythonie mamy dwie instrukcje iteracyjne: **while** i **for**.

Instrukcja **while** ma schemat:

while (test):
kod do wykonania

While możemy przetłumaczyć jako *dopóki*, czyli dopóki test daje prawdę powtarzaj wykonywanie kodu. Graficznie można to przedstawić jako,



Odpowiednikiem tego rysunku jest kod,

```
1 i=1
2 while i<5:
3     print('Cześć')
4     i=i+1
```

Cześć
Cześć
Cześć
Cześć

Wracając do poprzedniego zadania jego kod z zastosowaniem instrukcji **while** przyjmie postać,

```
1 suma=0
2 i=1
3 while suma<=100:
4     suma=suma+i
5     print(i,suma)
6     i=i+1
```

Blok 4-6 linia jest powtarzany dopóki suma jest mniejsza niż 100. Gdyby wartość sumy nie zmieniała się, blok byłby wykonywany bez końca. I program musiałby być przerwany przez zamknięcie środowiska w którym jest wykonywany. Istnieje instrukcja **break**, która może być użyta do przerywania pętli. Sposób jej użycia został pokazany poniżej,


```

1 suma=0
2 i=1
3 while suma<=100:
4     suma=suma+i
5     if suma>100:
6         break
7     print(i,suma)
8     i=i+1

```

Różnica w działaniu obu programów polega na tym, że program bez `break` drukuje wartość sumy większej od 100 i dopiero wtedy ją testuje i kończy iterację. Użycie `break` powoduje przerwanie iteracji przed drukiem tej wartości.

Przykład 15

Napisz 20 liczb tworzonych jako $i \cdot 10$ (gdzie i liczba 1,2,3....)

Rozwiązanie

Iteracja za pomocą `while` jest podstawą jednej z bardziej efektywnych metod poszukiwania nieznanych wartości. Niech naszym zadaniem będzie napisanie programu do znalezienia pierwiastka kwadratowego dowolnej liczby całkowitej. Chcemy ją wyznaczyć z dokładnością co najmniej do 0.1.

Jeżeli np. chcemy wyznaczyć pierwiastek liczby 7541 to sprawdzimy wszystkie liczby rzeczywiste od 1 do 7541 z krokiem 0.0001 czyli 1, 1.0001, 1.0002 7540.9999. Zadanie tylko z pozoru wygląda na beznadziejne, wykorzystamy ogromną prędkość działania komputera. Metoda ta nazywa się **brute force** i ma jedną niewątpliwą zaletę jaką jest jej prostota. Program będzie miał postać,

```

1 liczbaC=7541
2 dokl=0.1
3
4 p_liczbaC=1
5
6 while abs(liczbaC-p_liczbaC**2)>dokl:
7     p_liczbaC=p_liczbaC+dokl/100
8     if p_liczbaC>liczbaC:
9         print('Brak sukcesu')
10        break
11 print(liczbaC,round(p_liczbaC,3),round(p_liczbaC**2,3))

```

7541 86.839 7541.012

Problem rozwiązujemy za pomocą zgadywania wartości tego pierwiastka. Wiemy, że musi być to jakaś wartość pomiędzy 1 a 7541. Sprawdzamy je po kolei z krokiem 0.01/100 czyli 0.0001. Łatwo policzyć, że do sprawdzenia mamy 75410000 wartości. Zaczynamy od wartości 1, czyli pierwszym kandydatem na pierwiastek jest wartość 1 (`p_liczbaC`). Następnie sprawdzamy czy różnica pomiędzy proponowaną wartością pierwiastka podniesioną do kwadratu różni się od 7541 o mniej niż dokładność (0.1). Ponieważ różnica może mieć znak plus, albo minus użyliśmy funkcji `abs` zwracającej wartość bezwzględna. Jeśli warunek jest spełniony mamy pierwiastek, jeśli nie, to zwiększamy proponowaną wartość o $0.1/100 = 0.001$ i powtórnie sprawdzamy. Powtarzamy to tak długo, aż znajdziemy pierwiastek albo osiągniemy wartość 7541, co oznacza, że nie osiągnęliśmy celu. Powinniśmy wtedy zwiększyć dokładność.

Przykład 16

Rzucamy monetą. Ile razy musimy rzucić aby 5 razy pod rząd otrzymać reszkę.

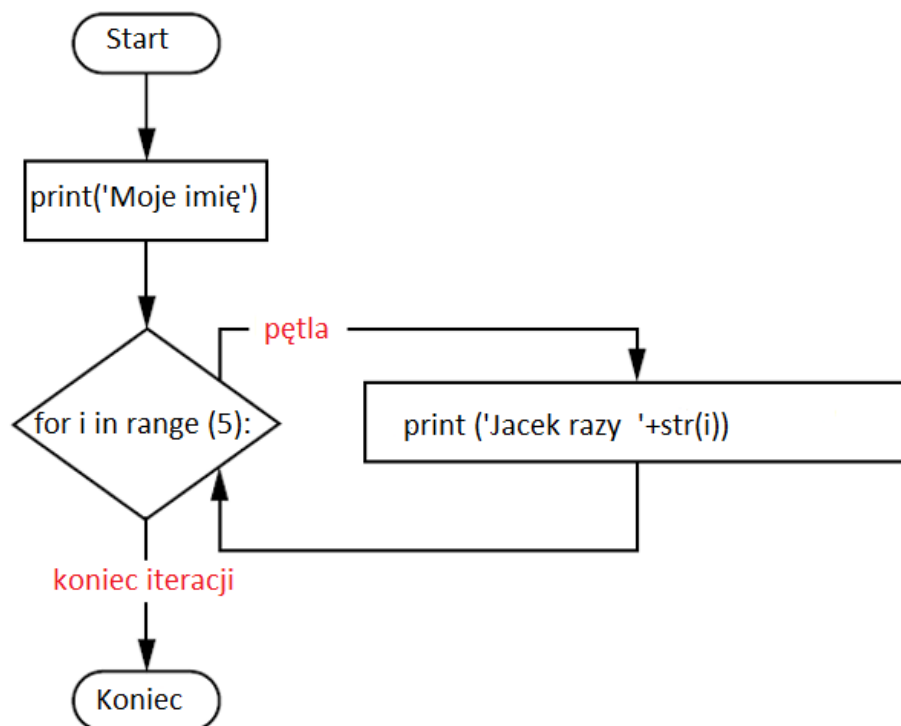
Rozwiązanie

Kolejną instrukcją służącą do iteracji jest instrukcja **for**, która ma schemat,

for i in range (....):

kod do wykonania

For możemy przetłumaczyć jako dla, następnie mamy zmienną sterującą (w naszym schemacie **i**) która zmienia się w kolejnych pętlach iteracji według wzorca określonego przez funkcję **range**. Graficznie instrukcję można przedstawić jako,



W instrukcji pętla będzie wykonana pięć razy, a zmienna sterująca przyjmie wartości 0,1,2,3,4.

Program będzie miał postać,

```
1 print('Moje imię')
2 for i in range(5):
3     print('Jacek razy '+str(i))
```

```
Moje imię
Jacek razy 0
Jacek razy 1
Jacek razy 2
Jacek razy 3
Jacek razy 4
```

Istnieją trzy podstawowe sposoby wprowadzania zmiennej sterującej w instrukcji **for** ,

```

1 for i in range(10):
2     print(i,end=',')
3 print()
4 for i in range(4,10):
5     print(i,end=',')
6 print()
7 for i in range(4,10,2):
8     print(i,end=',')

```

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

W pierwszym przykładzie jedna liczba oznacza, że zmienna sterująca przyjmuje wartości od 0 do tej liczby pomniejszonej o jeden (0....9). W trzecim, że zmienna sterująca przyjmowana jest ze skokiem odpowiadającym trzeciej liczbie w funkcji range.

Przykład 17

Rzucamy 1000 razy monetą, ile razy 3 razy z rzędu otrzymamy reszkę. Oblicz średnią wartość ze 100 takich prób.

[Rozwiązanie](#)

Przykład 18

Zbuduj tabliczkę mnożenia do 100 w postaci,

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

[Rozwiązanie](#)

Przykład 19

Wyrażenie na zamianę stopni Fahrenheita na stopnie Celsjusza ma postać:

$$C = \frac{5}{9}(F - 32)$$

Utworzyć tabelę konwersji stopni F na C od wartości 0 F do 100 F z krokiem 5 stopni F.

[Rozwiązanie](#)

Listy

Do tej pory większość typów zmiennych zawierało jedną wartość, wyjątkiem był tekst, którego najmniejszym elementem był pojedynczy znak. Tekst może zawierać wiele znaków, a dostęp do nich jest możliwy za pomocą indeksów.

Podstawowym typem danych, który może zawierać wiele różnych elementów jest **LISTA**. Jej schemat może być przedstawiony jako zmienna składająca się z wielu elementów, do których dostęp odbywa się podobnie jak w tekście za pomocą indeksów. Przy czym elementami mogą być wartości dowolnego typu, łącznie z listą.

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
------	------	------	------	------	------

Mamy tutaj listę C, która zawiera 6 elementów. Indeks pierwszego elementu ma wartość równą zero. Program definiujący taką listę, a następnie z niej korzystający wyglądałby tak,

```
1 C=[1.2,7,'punkt',234,-2,0]
2 print(C)
3 for i in range(6): print(C[i],end=',')
```

```
[1.2, 7, 'punkt', 234, -2, 0]
1.2,7,punkt,234,-2,0,
```

Linia 1 definiuje listę C, proszę zwrócić uwagę, że używamy do jej zdefiniowania nawiasy kwadratowe, a poszczególne jej elementy są oddzielone przecinkiem. Linia 2 do wydruk listy, a 3 wydruk wszystkich jej elementów po kolei. Poniżej przedstawiono trzy podstawowe sposoby tworzenia list,

```
1 lista1=[1,2,3,'punkt',12.2]
2 lista2=[]
3 lista3=[0]*10
```

```
1 print(lista1)
2 print(lista2)
3 print(lista3)
```

```
[1, 2, 3, 'punkt', 12.2]
[]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

W 2 linii tworzona jest lista pusta a w trzeciej dziesięcioelementowa wypełniona określoną wartością. Za pomocą indeksów mamy dostęp do dowolnego elementu listy, możemy także zmieniać jego wartość.

```

1 print(lista1)
2 print(lista1[0], lista1[3])
3 lista1[2] = 3.23
4 print(lista1)

```

```

[1, 2, 3, 'punkt', 12.2]
1 punkt
[1, 2, 3.23, 'punkt', 12.2]

```

Podstawowe operacje na listach to określenie liczby elementów (funkcja **len()**) oraz metody dodawania do końca listy (**.append()**) i zamiana tekstu na elementy listy (**.split()**). Zasady ich działania pokazano na przykładach poniżej,

```

1 print(len(lista1))
2 print(len(lista2))

```

```

5
0

```

```

1 for i in range(len(lista1)):
2     print(lista1[i])

```

```

1
2
3.23
punkt
12.2

```

Metody, które stosuje się do obiektów, umieszcza się po obiekcie (liście) poprzedzone kropką. Metoda **.append()** pozwala nam na dodawanie elementów do końca listy i w ten sposób po utworzeniu pustej listy jej budowanie.

```

1 print(lista2)
2 lista2.append(1); lista2.append(2.11); lista2.append('punkt')
3 for i in range(len(lista2)): print(lista2[i])

```

```

[]
1
2.11
punkt

```

Metoda **.split()** pozwala na rozbicie tekstu na elementy, definiując znak rozdzielający (np. przecinek).

```

1 sss='1,2,3.45,7'
2 lista4=sss.split(',')
3
4 for i in range(len(lista4)): print(lista4[i])

```

```

1
2
3.45
7

```

Istnieje wiele innych metod i funkcji związanych z listami. Korzystając z indeksów można także na listach przeprowadzać **slicing**. Bardzo podobną strukturą danych do listy jest **TUPLA**. Praktycznie jest to lista, w której nie można zmieniać raz utworzonych wartości. Tuplę tworzymy w taki sam sposób jak listę, ale używamy nawiasów półokrągłych. Nie obsługuje ona także metody `append()`. Dostęp do tupli odbywa się za pomocą indeksów z użyciem nawiasów kwadratowych, czyli tak jak w listach.

```
1 x=1;y=2
2 tupla2=(x,y)
3 print(tupla2)
4 print(tupla2[0],tupla2[1])
```

```
(1, 2)
1 2
```

Przykład 20

Utwórz listę zawierającą ciąg liczb nieparzystych od 1 do 99.

[Rozwiązanie](#)

Przykład 21

Utwórz listę 100 losowych współrzędnych punktów (na obszarze określonym przez $x = -100.0$ do 100 ; $y = -100$ do 100). Poszczególne współrzędne powinny być zapisane jako tuple.

[Rozwiązanie](#)

Przykład 22

Wykorzystaj listę z zadania 21 do znalezienia najmniejszej i największej odległości pomiędzy punktami.

[Rozwiązanie](#)

Tablice

Listy są bardzo elastyczną strukturą danych, ale ceną za to są ich dwie istotne wady. Operacje na listach są dość powolne i jest to z założenia struktura jednowymiarowa. Odpowiedzią na potrzebę szybkich operacji w n-wymiarowej tablicy stało się powstanie modułu **NumPy**. Jest to jedna z najważniejszych bibliotek współczesnego Pythona służąca do przetwarzania danych. Jest ona też fundamentem, na którym powstało wiele powszechnie używanych bibliotek obliczeniowych. **NumPy** daje możliwość tworzenia tablic o dowolnych wymiarach, ale zwykle korzysta się z 1-wymiarowych (wektorów) i 2-wymiarowych. Wszystkie elementy tablicy są jednego typu. Pierwszym krokiem jest zaimportowanie modułu. Zwyczajowo wykorzystuje się inny sposób importowania w postaci,

```
1 import numpy as np
```

Ten sposób importowania, wymaga, aby każda funkcja modułu była poprzedzona przez **np**. Typowe tablice numpy mają postać,

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
------	------	------	------	------	------

C[0,0]	C[0,1]	C[0,2]	C[0,3]	C[0,4]	C[0,5]
C[1,0]	C[1,1]	C[1,2]	C[1,3]	C[1,4]	C[1,5]
C[2,0]	C[2,1]	C[2,2]	C[2,3]	C[2,4]	C[2,5]
C[3,0]	C[3,1]	C[3,2]	C[3,3]	C[4,4]	C[5,5]

Górna część rysunku przedstawia tablicę 1-wymiarową (wektor). Jej struktura jest taka sama jak listy. Tablica 2-wymiarowa składa się z wierszy i kolumn. Komórki tablicy są opisane przez indeksy. Pierwszy indeks opisuje numer wiersza, a drugi kolumny. Podobnie jak w tablicach indeksy przyjmują wartość od zera. W tablicy 2-wymiarowej komórka o indeksach 0,0 znajduje się w lewym górnym rogu.

Wektor (tablica 1-wymiarowa) jest tworzony za pomocą funkcji **np.zeros** która zawiera rozmiar wektora i jego typ (float albo int).

```
1 vect1=np.zeros(10,float)
2 vect2=np.zeros(6,int)
3
4 print(vect1)
5 print(vect2)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0 0 0 0 0 0]
```

Podobnie jak w liście dostęp do poszczególnych elementów odbywa się za pomocą indeksów, rozmiar tablicy, można uzyskać za pomocą metody **shape**, która zwraca n-wymiarową tuplę.

```
1 print(vect1.shape)
2 rozmiar=vect1.shape
3 print(rozmiar[0],type(rozmiar[0]))
4 for i in range(rozmiar[0]):print(vect1[i],end=',')
```

```
(10,)
10 <class 'int'>
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
```

Poniżej przedstawiono program wypełniający tablicę losowymi wartościami całkowitymi,

```
1 from random import *
2
3 rozmiar=vect2.shape
4 for i in range(rozmiar[0]):
5     vect2[i]=randint(0,100)
6
7 print(vect2)
```

```
[52 16 50 90 78 23]
```

Nazwa zmiennej jest powiązana z tablicą. Tablica może mieć parę nazw (klonowanie), ale wskazują one jedną tablicę. Pokazuje to następujący kod,

```
1 vect2A=vect2
2 print(vect2)
3 print(vect2A)

[ 0 71 72 82 27 97]
[ 0 71 72 82 27 97]
```

```
1 vect2.sort()
2 print(vect2)
3 print(vect2A)

[ 0 27 71 72 82 97]
[ 0 27 71 72 82 97]
```

Przypisanie do vect2A wektora vect2 nosi nazwę **klonowania**. Drukując każdą ze zmiennych, odnosimy się do jednej istniejącej tablicy (wektora). Posortowanie zmiennej vect2 za pomocą metody **.sort()** sortuje tablicę. Drukowanie zmiennej vect2 i vect2A da ten sam efekt, bo tablica jest jedna. Jeżeli chcemy, aby każda zmienna posiadała własną tablicę należy użyć metody kopiowania **.copy()**.

```
1 vect2A=vect2.copy()
2 print(vect2)
3 print(vect2A)

[32  4 31 50 64  9]
[32  4 31 50 64  9]
```

```
1 vect2.sort()
2 print(vect2)
3 print(vect2A)

[ 4  9 31 32 50 64]
[32  4 31 50 64  9]
```

Kopiowanie powoduje powstanie niezależnej kopii tablicy i sortowanie jednej z nich nie będzie miało wpływu na drugą. Zasada klonowania i kopiowania dotyczy także list.

Tablice 2- wymiarowe są tworzone w podobny sposób, ale zawierają dwa parametry w nawiasach, które definiują ich rozmiar. Pierwszy parametr określa liczbę wierszy, a drugi liczbę kolumn. Podobnie jak w tablicach jednowymiarowych, możemy tworzyć tablice typu rzeczywistego (float) i całkowitego (int).


```

1 tab1=np.zeros((10,5),float)
2 tab2=np.zeros((6,3),int)
3
4 print(tab1)
5 print(tab2)

```

```

[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]

```

Do przypisywania nowych wartości używa się indeksów w nawiasach kwadratowych. Rozmiar tablicy możemy określić za pomocą metody **shape**, która zwraca tuplę z liczbą wierszy i kolumn.

```

1 # INDEKSY pierwszy-wiersz, drugi-kolumny
2 tab1[2,3]=7.2
3 print(tab1[2,3])
4 print(tab1.shape)
5 print(f'wiersze = {tab1.shape[0]}, kolumny = {tab1.shape[1]}')

```

```

7.2
(10, 5)
wiersze = 10, kolumny = 5

```

Przykład 23

Utwórz tablicę 12x12, pierwsze cztery wiersze mają zawierać 0, następne cztery liczbę 1, a ostatnie cztery liczbę 2.

[Rozwiązanie](#)

Przykład 24

Utwórz tablicę 12x12, która zawiera rzeczywiste liczby losowe w zakresie 0 - 10.

[Rozwiązanie](#)

Przykład 25

Utwórz tablicę o rozmiarze 10 wierszy i 19 kolumn i wypełnij ją zerami i jedynkami, tak aby jedynki tworzyły symetryczny trójkąt równoramienny o podstawie w dolnej części tablicy i wierzchołku w górnej.

[Rozwiązanie](#)

Dla tablic można obliczać statystyki za pomocą szeregu metod. Poniżej przedstawiono niektóre z nich.

```
1 print(tab3.mean())
2 print(tab3.min())
3 print(tab3.max())
4 print(tab3.std())
5 print(tab3.sum())
```

```
5.118967800976601
0.05769683797325009
9.939043502027863
2.63303446048883
737.1313633406305
```

Przykład 26

Napisz program do którego wprowadza się dowolny zbiór liczb (ciąg liczb oddzielonych przecinkiem). Program wyprowadza liczbę maksymalną i informację ile razy wystąpiła, oraz liczbę minimalną wraz z ilością jej wystąpień.

[Rozwiązanie](#)

Czytanie i zapisywanie plików tekstowych

Programy bardzo często służą do przetwarzania danych. W modelowym przypadku będzie to polegało na wczytaniu do programu danych z pliku tekstowego, ich przetworzeniu i zapisaniu rezultatów do nowego pliku tekstowego. Typowy plik tekstowy został przedstawiony poniżej,

```
3.21,7.2,100
4,2.14,202
18.1,123.4,564.2
2.3,11,112
9,8,8
```

Plik składa się z linii, a poszczególne wartości są w nim porozdzielane wybranym znakiem (najczęściej jest to przecinek lub średnik). Jest to najprostsza forma zapisu danych. Pliki te mają najczęściej rozszerzenie .txt lub .csv (od Comma Separated Values). Dostęp do plików, oraz określenie miejsca w którym zostaną zapisane wymaga podania ścieżki i ich nazwy. Istnieją trzy sposoby formatowania tej ścieżki:

```
path = ' C:\\katalog\\projekt\\dane\\plik.txt'
```

```
path = ' C:/katalog/projekt/dane/plik.txt'
```

```
path = r' C:\katalog\projekt\dane\plik.txt'
```

Ze względu na to, że kopiowana ścieżka ma postać ostatniego przykładu, najłatwiej będzie go stosować. Proszę zwrócić uwagę, że string jest poprzedzony literą **r**. Sama ścieżka jest tekstem i tak jak tekst może być otoczona apostrofem pojedynczym lub podwójnym.

Czytanie pliku może odbywać się za pomocą schematu,

```
in_plik = open( ścieżka + nazwa pliku,"r")
for line in in_plik.readlines():

    lista=line.split(",")

in_plik.close()
```

W pierwszej linii otwieramy plik do czytania za pomocą funkcji **open** (pierwszym argumentem jest pełna ścieżka drugim litera "r". Plik powiązany jest rezultacie otwarcia ze zmienną (tu in_plik). Następnie w pętli za pomocą metody **readlines** czytane są linie po linii jako stringi. Tekst tych linii jest rozdzielany na tekstowe elementy listy, za pomocą metody **split** . Ostatnia linia zamyka plik. W praktyce przykładowy program będzie miał postać,

```
1 in_plik=open(r"dane/in1.txt","r")
2
3 for line in in_plik.readlines():
4     lista=line.split(",")
5     x1=float(lista[0])
6     x2=float(lista[1])
7     x3=float(lista[2])
8     print(x1,x2,x3)
9 in_plik.close()
```

Jeżeli program w pythonie lub jupyter notebook znajduje się danym katalogu to ścieżka może zaczynać się od następnego katalogu (w przykładowych ścieżkach powyżej, program znajdowałby się w katalogu projekt).

Tworzenie pliku i jego zapisywanie będzie miało schemat,

```
out_plik = open( ścieżka + nazwa pliku,"w")
out_plik.write(linia tekstu)

out_plik.close()
```

Pierwsza linia jest identyczna, zawiera natomiast literę "w" . Po otwarciu nowego pliku do zapisywania za pomocą metody **write** wprowadzamy tekst (string). Jeżeli chcemy aby nastąpiła zmiana linii musimy wprowadzić specjalny znak "\n" (znak nowej linii). Może on być dodany do stringu na końcu, albo oddzielnie. Kod poniżej drukuje tabliczkę mnożenia i jednocześnie zapisuje ją do pliku. Można wykonać go kilkakrotnie, bo za każdym razem plik zostanie utworzony na nowo. Przy drukowaniu użyto funkcji **print** w postaci **print(f'{i*j:4}',end="")** co powoduje, że po jej wykonaniu nie następuje przejście do nowej linii, ale kolejny **print(...)**, będzie kontynuacją wydruku. Aby przejść do nowej linii wykorzystano funkcję **print()**.

Zapis wykorzystuje metodę **write**, która po wykonaniu nie przechodzi do następnej linii, jeśli na końcu tekstu nie zastosowano znaku nowej linii. Jest on zapisywany oddzielnie po zakończeniu wiersza.

```

1 out_plik=open(r"dane/tab_mnoz.txt","w")
2
3 for i in range(1,11):
4     for j in range(1,11):
5         print(f'{i*j:4}',end='')
6         out_plik.write(f'{i*j:4}')
7     print()
8     out_plik.write('\n')
9 out_plik.close()

```

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Przebieg typowego procesu polegającego na odczytaniu pliku tekstowego z danymi, ich przekształceniu i zapisaniu do nowego pliku może się odbywać według dwóch schematów. W pierwszym proces czytania i zapisywania jest prowadzony równocześnie linia po linii.

```

1 # Wersja pierwsza - wiersz po wierszu
2
3 in_plik=open(r"dane/in1.txt","r")
4 out_plik=open(r"dane/out1.csv","w")
5
6 for line in in_plik.readlines():
7     lista=line.split(",")
8     x1=float(lista[0])
9     x2=float(lista[1])
10    x3=float(lista[2])
11    x2=round(x2*2,3)
12    x3=round(x3*3,3)
13    sss=f"{x1},{x2},{x3}\n"
14    out_plik.write(sss)
15
16 in_plik.close()
17 out_plik.close()

```

Najpierw otwieramy plik do czytania i zapisu. Następnie czytamy linię i odzyskujemy z niej dane (do x1,x2,x3), zmieniamy ich wartości i zapisujemy nową linię do pliku out_plik.

Drugi sposób polega na realizacji zadania w trzech etapach. W etapie pierwszym tworzymy tablicę o liczbie wierszy przekraczającej rozmiar danych i wczytujemy do niej dane. W etapie drugim przetwarzamy tablicę, a w trzecim wpisujemy jej zawartość do nowego pliku. Wprowadzamy także zmienną (linia 9), która służy do zliczania liczby wierszy z danymi w tablicy. Cały proces jest pokazany jest na przykładzie poniżej.

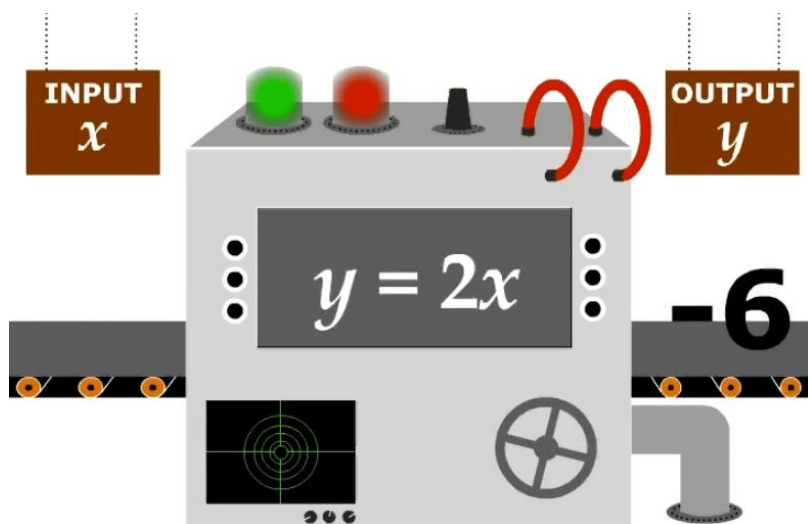
```

1  #Wersja druga - czytamy do tablicy, przetwarzamy tablicę, zapisujemy z tablicy
2  import numpy as np
3
4  xxx=np.zeros((30,3),float)
5  in_plik=open(r"dane/in1.txt","r")
6  out_plik=open(r"dane/out2.csv","w")
7
8  # wczytanie do tablicy
9  ii=0
10 for line in in_plik.readlines():
11     lista=line.split(",")
12     x1=float(lista[0])
13     x2=float(lista[1])
14     x3=float(lista[2])
15     xxx[ii,0]=x1;xxx[ii,1]=x2;xxx[ii,2]=x3
16     ii=ii+1
17 in_plik.close()
18
19 # przetworzenie tablicy
20 for i in range(ii):
21     xxx[i,1]=xxx[i,1]*2
22     xxx[i,2]=xxx[i,2]*3
23
24 # zapisanie tablicy
25 for i in range(ii):
26     sss=f"{round(xxx[i,0],2)},{round(xxx[i,1],2)},{round(xxx[i,2],2)}\n"
27     out_plik.write(sss)
28 out_plik.close()

```

6. Funkcje

Funkcje w Pythonie to autonomiczne bloki programu, które mogą być wykonywane przez ich wywołanie. Istotnymi ich elementami jest zbiór parametrów funkcji, czyli danych które wprowadzamy (na rysunku INPUT np. -3). Następnie funkcja podejmuje działanie ($y=2*x$) i zwraca wartość (na rysunku OUTPUT -6).



Funkcje mogą być rozumiane jako skrzynki do których wprowadzamy wartości i otrzymujemy nowe wartości, po przekształceniu ich przez funkcję. To co wykonuje musi być określone, ale w jaki sposób to robi już nie. Wnętrze funkcji jest niewidoczne w trakcie jej wykorzystywania. Każda funkcja ma swoją **NAZWĘ** i **ZBIÓR PARAMETRÓW** (mogą być opcjonalne). Szereg

funkcji jest wbudowanych w podstawowy Python. Na przykład funkcja **round** , która służy do zaokrąglania liczb rzeczywistych jest zademonstrowana poniżej na przykładzie,

```
1 print(type(round))
2
3 x=45677.123456789
4 y1=round(x)
5 y2=round(x,2)
6
7
8 print(y1,y2)
```

```
<class 'builtin_function_or_method'>
45677 45677.12
```

Funkcja może mieć jeden, albo dwa parametry. Przy jednym parametrze zwraca liczbę całkowitą, przy dwóch (liczba miejsc po kropce dziesiętnej) liczbę rzeczywistą zaokrągloną matematycznie. Istnieje funkcja **help** , która umożliwia otrzymanie opisu każdej funkcji.

```
1 print(help(round))
```

Help on built-in function round in module builtins:

```
round(...)
    round(number[, ndigits]) -> number
```

```
    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument, otherwise the
    same type as the number. ndigits may be negative.
```

None

Z opisu możemy się dowiedzieć, że drugi parametr może mieć wartość ujemną, jeśli sprawdzimy jak to działa otrzymamy,

```
1
2 x=45677.123456789
3
4 y2=round(x, -2)
5 print(y2)
```

```
45700.0
```

Najprostsza metoda zdefiniowania własnej funkcji polega na wykorzystaniu schematu,

```
def nazwa(argument1,argument2):
    linia 1
    .....
    linia n
    return wynik
```

natomiast jej wywołanie będzie miało postać: `x = nazwa(zmienna1, zmienna2)` . Pierwsza linia zaczyna się słowem kluczowym **def** następnie umieszczamy nazwę funkcji i w nawiasie jej argumenty. Zmienne wprowadzone w definicji funkcji występują w bloku jej kodu zaznaczonego wcięciem względem linii z definicją. Nazwy argumentów (zmienne) są używane wewnątrz bloku i mają charakter lokalny. Istnieją tylko wewnątrz funkcji. Podstawy działania funkcji można prześledzić na kodzie poniżej,

```
1 def moja_funkcja(a):
2     b=a*a*a
3     return b
4
5 print(moja_funkcja(2))
6 z=6
7 print(moja_funkcja(z))
8 y=moja_funkcja(z)
9 print('y= ',y)
10 print(b)
```

```
8
216
y= 216
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-3-37d608329d0a> in <module>
      8 y=moja_funkcja(z)
      9 print('y= ',y)
----> 10 print(b)
```

```
NameError: name 'b' is not defined
```

Wykonanie pierwszych 3 linii kodu definiuje funkcję i czyni ją dostępną dla programu. Argument funkcji został nazwany a (jego typ wynika ze sposobu jego wykorzystania wewnątrz funkcji), i wraz ze zmienną b mają charakter lokalny. Próba ich wykorzystania poza funkcją spowoduje błąd. Funkcja może być wykorzystana na różne sposoby. Podstawowy sposób to przypisanie wartości, którą funkcja zwraca, do zmiennej. Funkcja może nie zwracać zmiennej czyli nie posiadać linii return.

```
1 def UWAGA(nr):
2     print('UWAGA nr '+str(nr))
3
4 UWAGA(4)
```

```
UWAGA nr 4
```

Charakterystyczną cechą funkcji jest tworzenie **lokalnej przestrzeni** dla zmiennych. Argumenty i zmienne, które pojawiają się w funkcji (z lewej strony instrukcji przypisania) mają wyłącznie lokalny charakter i nie istnieją poza funkcją. To zagadnienie jest przedstawione na przykładzie poniżej,

```

1 def F(x):
2     global aa      # deklaracja że aa będzie globalne
3     aa=aa+1
4     y=x+bb          # użycie bb z programu, y nowa zmienna lokalna
5     x=x+y+aa
6     print('FF  x=',x, 'y=',y)
7     return x
8
9 aa=5;bb=4
10 x=3;y=2
11 print('PP  x=',x, 'y=',y)
12 z=F(x)
13 print('PP  x=',x, 'y=',y , 'z=',z)
14 print(aa)

```

```

PP  x= 3 y= 2
FF  x= 16 y= 7
PP  x= 3 y= 2 z= 16
6

```

W programie głównym mamy zmienną aa. Funkcja F(x) modyfikuje jej wartość (linia 3). Aby to było możliwe zmienna ta musi być zadeklarowana jako **global** (linia 2). Zmienna x i y występuje zarówno w programie głównym jak funkcji. Są to jednak zupełnie różne zmienne występujące w dwóch różnych przestrzeniach i nie mają ze sobą nic wspólnego. W programie głównym występuje też zmienna bb, jest ona wykorzystywana w funkcji, bo pojawia się tylko z prawej strony.

Przykład 27

Utwórz funkcję której parametrem jest tablica jednowymiarowa i która zwraca liczbę wartości w tablicy, które są mniejsze od zera..

Rozwiązanie

Przykład 28

Utwórz funkcję o dwóch parametrach x1 i x2. Funkcja zwraca 1 (jeśli $x_1 > x_2$), 2 (jeśli $x_1 < x_2$) albo 0 (jeśli $x_1 = x_2$).

Rozwiązanie