Struktura programu

Tabulator = 4 spacje

Dozwolone są spacje pomiędzy operatorami, np.: bound = x * 12 + 13, print (x + 2)

oraz spacje za przecinkami/średnikami, np.: print(“11”, ”12”). Nie wpływają one na działanie kodu, a pozwalają na zachowanie „czystości kodu”

Stałe użycie cudzysłowu, tzn.: ‘hello’ lub ”hey”. Należy zachować jedną konwencję w kodzie.

#komentarz zajmujący 1 linię

”””

Komentarz zajmując wiele linii. Treść zapisana w komentarzu nie jest przetwarzana. Nie wpływa na działanie kodu.

Komentarz wieloliniowy można zapisać również używając pojedynczych cudzysłowów (‘’’komentarz‘’’)

”””

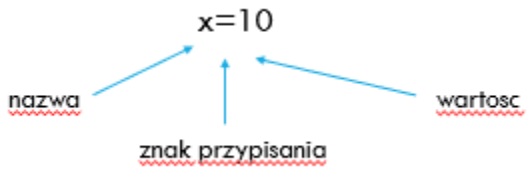
Przykładowy program:

```
print(“Ten program dodaje 2 liczby”)
num1=input(“Wpisz pierwsza liczbę: “)
num1=int(num1)
num2=input(“Wpisz druga liczbę: “)
num2=int(num2)
total=num1+num2
print(“Suma liczb to “ +str(total) +”.”)
```

Zmienne

Zmienna to miejsce do przechowywania informacji w programie

- każda zmienna posiada **nazwę** oraz **wartość**
- można stworzyć nowa zmienna poprzez przypisanie wartości



- można zmienić wartość zmiennej poprzez nowe przypisanie x=5
- można ustawić wartość zmiennej za pomocą działań matematycznych x=5+2

Przypisanie

Aby przypisać wartość używamy znaku równości (=)

- zmienna jest tworzona przy pierwszym przypisaniu do niej wartości
- kolejne przypisania powodują zmianę wartości zmiennej

UWAGA!!! przypisanie nie oznacza matematycznej równości!!

Przykład:
total=total+1

- przypisanie: najpierw rozwiązuje prawa stronę, następnie przypisuje wartość do lewej strony

  total=3

  total=3+1

- zmienne są widoczne tylko wewnątrz funkcji, w której są stworzone (“scope”)
  - Przykład: jest zmienna jest stworzona w funkcji main(), jest ona widoczna tylko w funkcji main()

Nazwy zmiennych

zmienna **MUSI** mieć nazwę i spełniać następujące warunki:

- zaczynać się litera lub od podkreślenia (_)
- nie może być wbudowana komenda Pythona (np.: for, if)
- nie powinna być użyta wbudowana funkcja (np.: int, float) [miękka zasada]
- może zawierać tylko litery, cyfry lub podkreślenia

UWAGA!!! ma znaczenie czy nazwa zaczyna się z małej czy z wielkiej litery

- **Zmienna** oraz **zmienna** to dwie różne nazwy

zmienne powinny:

- być opisowe (x nie jest dobrą nazwą, chyba że podajemy koordynaty geograficzne)
- być w formacie snake case: jakas_zmienna

Typy zmiennych

Kiedy przechowujemy jaką informację w zmiennej, staje się ona obiektem Pythona

Obiekty mogą być różnych typów i rozmiarów

integer – liczby całkowite, np.: 400, 2, -5	**int**
float – liczby rzeczywiste, np.: 400.0, 2.0, -5.0	**float**
string – dane tekstowe, np.: “hello”, ‘10’	**str**
boolean – wartości logiczne (True/False)	**bool**

Zmienna typu float nie ma jasno zdefiniowanej „następnej liczby”, np.: po 1.5 może być 1.51 lub 1.501 lub 1.5001, itd…

Zmienna typu int ma zdefiniowaną kolejność liczb, np.: 1, 2, 3…

print()

Funkcja print drukuje tekst w terminalu.

Można użyć kilku komend print, żeby wydrukować kilka linii tekstu:

```
print(„jeden”)
print(„dwa”)
print(„trzy”)
…
…
print(„n”)
```

```
print(“jeden”, “dwa”, “trzy”)
```

Można użyć kilku wersji printa:

```
liczba_rzeczywista = 17.3
liczba_calkowita = 580
tekst = 'chomik'

print(tekst, 'ma', liczba_rzeczywista , 'lat i waży', liczba_calkowita, 'gram’)                    #1


print(tekst +'ma' +str(liczba_rzeczywista ) +'lat i waży' +str(liczba_calkowita) +'gram')         #2
print(tekst +' ma ' +str(liczba_rzeczywista ) +' lat i waży ' +str(liczba_calkowita) +' gram ‘)   #3


print(f"{tekst} ma {liczba_rzeczywista } lat i waży {liczba_calkowita} gram")                     #4
print(f’{tekst} ma {liczba_rzeczywista } lat i waży {liczba_calkowita} gram’)                      #5
```

outputy:

1 chomik ma 17.3 lat i waży 580 gram

2 chomikma17.3lat i waży580gram

3 chomik ma 17.3 lat i waży 580 gram

4 chomik ma 17.3 lat i waży 580 gram

5 chomik ma 17.3 lat i waży 580 gram

input()

Funkcja input dostaje dane od użytkownika

Drukuje tekst podany w cudzysłowach “” lub ‘’

- Czeka na odpowiedz użytkownika
- W tym wypadku input jest przypisany do zmiennej (num1)
- Wprowadzone dane są typu tekstowego nawet jeżeli użytkownik wprowadzi numer

Przykład:

num1=input(‘Wpisz pierwsza liczbę: ‘)

Łączenie zmiennych typu *string*

Operator + powoduje połączenie (concantention) zmiennych typu string

```
Str1="co"
Str2=" "
Str3="slychac"
Str4=Str1+Str2+Str3
print(Str4)
```

Jest możliwe łączenie zmiennych tego samego typu, ale nie różnego

- **total** było typu int, dlatego trzeba go zamienić na string
- **UWAGA! Oryginalna wartość total to wciąż int**

Definiowanie typu zmiennej podczas *input()*

Zmienne można również definiować podczas pytania o input()

```
num1=int(input("Wpisz pierwsza liczbe: "))
```

lub

```
num1=float(input("Wpisz pierwsza liczbe: ))
```

Znaki modyfikacji

\ Wstawiony przed znakiem specjalnym zmienia go w regularny znak

Przykład:

```
print('he\'s a boy')
print("\"hello\"")
```

\t tabulator;

\n nowa linia;

\r powrót karetki;

\b backspace;

Wyrażenia matematyczne

num1 = 7

num2 = 2

num3 = num1 + num2	+	dodawanie	num3=9
num3 = num1 - num2	-	odejmowanie	num3=5
num3 = num1 * num2	*	mnożenie	num3=14
num3 = num1 / num2	/	dzielenie	num3=3.5
num3 = num1 ** num2	**	potęgowanie	num3=49
num3 = num1 // num2	//	dzielenie (wynik int)	num3=3
num3 = num1 % num2	%	reszta z dzielenia	num3=1
num3= -num1	(-)	liczba przeciwna	num3=-7

Działania matematyczne są wykonywane w następującej kolejności:

- nawiasy okrągłe
- potęgowanie
- mnożenie I dzielenie (ten sam priorytet)
- dodawanie I odejmowanie (ten sam priorytet)

Działania o tym samym priorytecie są wykonywane od lewej do prawej strony

W razie wątpliwości lub błędów w wyniku sprawdź/dodaj nawiasy

Stałe

```
PI = 3.14

INCH_TO_CM = 2.54
```

Stałe pozwalają na zachowanie “czystości” kodu (są dobrze czytelne)

Opisowe nazwy, w formacie UPPER_SNAKE_CASE

Podobieństwo do zmiennych; nie zmieniają wartości podczas egzekucji kodu

Definiujemy stałe w sposób ogólny, żeby wciąż działały w przypadku, gdy zmienimy wartość stałej

Przykład:

```
INCH_TO_CM = 2.54


inch = float(input(‘Podaj liczbę cali’))

cm = inch * INCH_TO_CM

print(inch, ‘cali to’, cm, ‘centymetrów’)
```

Biblioteki w Pythonie

Zbiór kodów do wielokrotnego wykorzystania

Biblioteki zawierają wbudowane m. in. funkcje, stałe, klasy pozwalające na natychmiastowe zastosowanie bez konieczności wyprowadzania skomplikowanych wzorów

https://docs.python.org/3.14/library/index.html

Aby zastosować bibliotekę w kodzie należy ją zaimportować. Bez importu, funkcje z biblioteki nie będą działały.

Implementacja biblioteki w kodzie: ↓ZASTOSOWANIE↓

import nazwa_biblioteki	(nazwa_biblioteki.funkcja)

aliasy - nazwy bibliotek można zastąpić inna nazwa w celu optymalizacji kodu, np.:

import numpy as np	(alias.funkcja)

z bibliotek można również importować pojedyncze elementy:

from nazwa_biblioteki import element	(funkcja)

Biblioteka *math*

```
import math
```

Wbudowane stałe:

math.pi	liczba pi
math.e	liczba eulera

Wbudowane funkcje:

math.sqrt(x)	zwraca pierwiastek kwadratowy liczby x
math.exp(x)	zwraca e^x

math.log(x) zwraca logarytm naturalny liczby x

wszystkie funkcje i stałe zawarte w bibliotece math:

https://docs.python.org/3/library/math.html

Struktura kodu stosując poznane elementy

import library_name	biblioteki
CONSTANTS	stałe
def main():	funkcje
zmienne	pozostałe
komendy	

wartości losowe

Sposób na wygenerowanie losowych wartości

Nie można uzyskać w technice komputerowej prawdziwej losowości, więc używamy liczb pseudolosowych

Aby zastosować wartości losowe, należy zaimportować bibliotekę random:

```
import random
```

Funkcja	Działanie
random.randint(min, max)	Zwraca losowy integer z przedziału od min do max, włącznie
random.random()	Zwraca losow float z przedziału od 0 do 1
random.uniform(min, max)	Zwraca losowy float z przedziału od min do max
random.seed(x)	Ustawia “seed” losowego generatora liczb jako x

Wyrażenia logiczne

To takie, które są prawdziwe lub fałszywe

Podczas stosowania wyrażeń logicznych używamy ==

wyrażenie zwraca wynik True lub False

UWAGA!!

== to nie to samo co =

True I False są wartościami typu boolean (bool)

Zwróćcie uwagę na wielką literę!!

Operatory porównania

==	porównanie (odniesienie do wartości)
!=	nierówność
>	większość
<	mniejszość
>=	większe lub równe
<=	mniejsze lub równe
is	jedna wartość jest tym samym co druga (odniesienie do obiektów)
is not	wartość nie jest tym samym co druga

Operatory logiczne

and x>2 and x<10

warunek spełniony tylko wtedy, gdy liczba jest z przedziału od 2 do 10

or x<2 or x>10

warunek spełniony wtedy, gdy liczba jest mniejsza od 2 lub większa od 10

not not(x>y)

warunek spełniony kiedy stwierdzenie jest prawdziwe, czyli jeśli x>y jest fałszywe

Instrukcje warunkowe

if warunek: <- uwaga na dwukropek

tab komenda **UWAGA!! Zwróćcie uwagę na wcięcie**

warunek musi być typu boolean (True/False)

warunek musi być prawdziwy, żeby została wykonana komenda

Przykład:

```
if x>y:
    print('x jest większe od y')
```

if-else

if warunek: <- uwaga na dwukropek

tab komenda

else: <- uwaga na dwukropek

tab inna komenda

Przykład:

```
if x>y:
    print('x jest większe od y')
else:
    print('x nie jest większe od y')
```

if-elif

if *warunek*:

tab komenda

Przykład:

```
if x==5:
    print('to jest cyfra 5')
```

```
elif inny_warunek:
    inna_komenda
elif jeszcze_inny_warunek:
    następna_komenda
```

```
elif x==3:
        print (‘to jest cyfra 3')
elif x==9:
        print (‘to jest cyfra 9')
```

if-elif-else

```
if warunek:
        komenda
elif inny_warunek:
        inna_komenda
else:
        następna_komenda
```

Przykład:

```
if x < y:
        print ('x jest mniejsza niż y')
elif x > y:
        print ('x jest większa niż y')
else:
        print ('x i y są równe ')
```

warunki zagnieżdżone

```
if warunek:
        komenda
else:
        if inny_warunek
                inna_komenda
        else:
                następna_komenda
```

Przykład:

```
if x == y:
        print ('x i y sa równe ‘)
else:
        if x < y:
                print ('x jest mniejsza niż y’)
        else:
                print ('x jest większa niż y')
```

pętla *while*

```
while warunek:
    komenda
```

```
x=0
while True:
    print(x)
    x += 1
```

break

- polecenie *break* powoduje wyjście z pętli nawet, jeśli warunek jest prawdziwy

Przykład:

```
x=0
while x<8:
    print(x)
    break
    x = += 1
```

continue

- przeskakuje iterację jeżeli trafi na określony warunek

Przykład:

```
x = 0
while x < 6:
    x += 1
    if x == 3:
        continue
print(x)
```

operatory przypisania

=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

pętla for

```
for x in range(0, 10):
    print(x)
```

różnica między pętlami *while* I *for*

- pętlę while używamy kiedy nie wiemy ile potrzeba iteracji do osiągnięcia oczekiwanego wyniku
- pętla for służy do wykonywania operacji kiedy znamy określoną liczbę iteracji

pętle zagnieżdżone

```
for x in range (0, 3):
    for y in range (0, 2):
        print(x, y)
```

for z krokiem

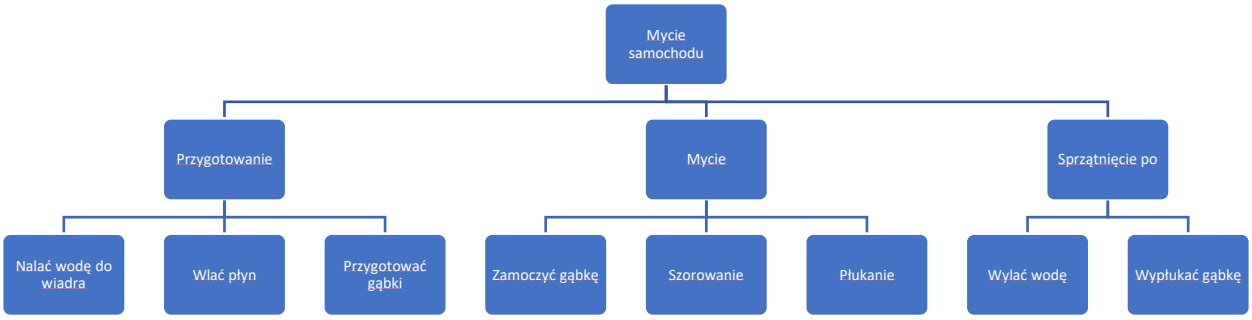
```
for x in range(0, 10, 2):
    print(x)
```

w jakich przypadkach używać pętli

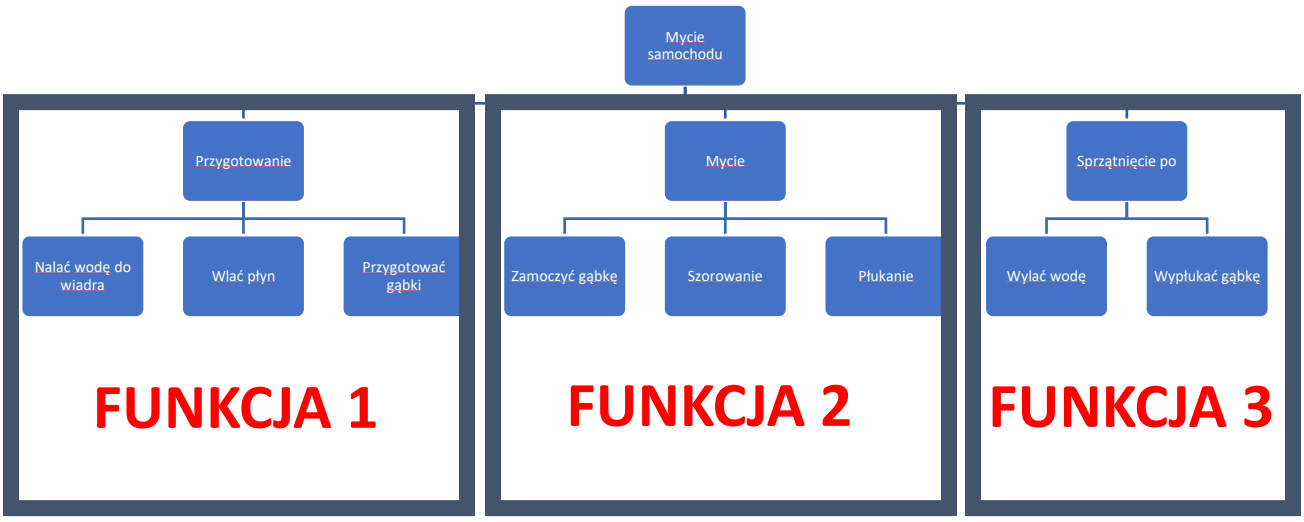
- przy małych datasetach
- w przypadku pracy na różnych typach danych
- operacje regex
- ekstrakcja stringów
- przy przetwarzaniu skomplikowanych struktur zagnieżdżonych
- jeżeli operacje nie mogą być zwektoryzowane

FUNKCJE

Dekompozycja polega na podzieleniu kodu na jak najmniejsze kroki.



Zbiór kroków nazywamy funkcją



funkcje wbudowane

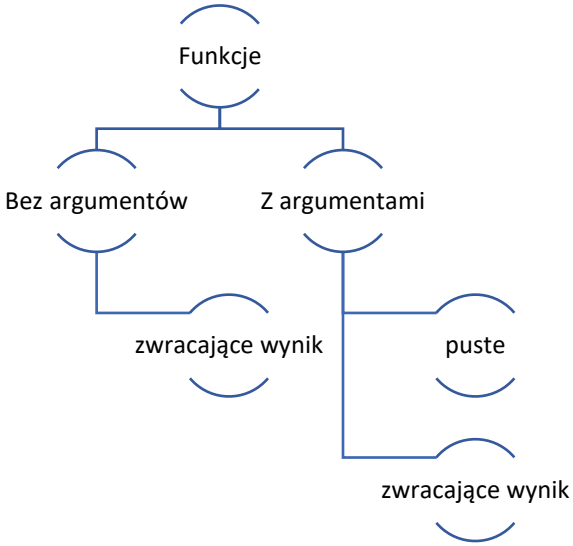
W Pythonie istnieją już funkcje wbudowane, które można tylko wywołać, bez potrzeby pisania ich na nowo.
Pewne funkcje wbudowane były już używane podczas zajęć:

print()
float()
str()
int()
len()
type()

Wszystkie funkcje wbudowane w Pythonie, ich opis oraz sposób wykorzystania można znaleźć na stronie z dokumentacją:

- https://docs.python.org/3/library/functions.html

rodzaje funkcji



definiowanie funkcji

W Pythonie można zdefiniować własne, dowolne funkcje

def *nazwa_funkcji()*: ← UWAGA!! Dwukropek

zawartość
zawartość
zawartość

W przypadku funkcji bez argumentów nawias powinien być pusty

Przykład:

```
def wydruk():
    print('Jest super…')
    print('…albo i nie')
```

Niektóre funkcje wymagają jednak argumentów (np.: type(zmienna)
Funkcje mogą także zwracać jakąś wartość/wynik lub nie (funkcja pusta)
Przykład:

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

def main():
  total = dodawanie(4, 5)
  print(total)

main()
```

nazwy funkcji

- nazwa funkcji może być dowolnym słowem
- lub ciągiem słów w formacie snake_case (słowa połączone podkreślnikiem)
- tak jak przy zmiennych, nazwa funkcji może zawierać litery, cyfry oraz niektóre znaki interpunkcyjne
- nazwa nie może zaczynać się od cyfry
- nie można używać słowo kluczowego jako nazwy funkcji
- nie powinno się używać tej samej nazwy funkcji i zmiennej

wywoływanie funkcji

Żeby funkcja zadziałała należy ją wywołać
Funkcje zdefiniowane samodzielnie, wywołujemy w taki sam sposób, co funkcje wbudowane

Przykład dla funkcji bez argumentów:

```
def wydruk():
   print('Jest super…')
   print('…albo i nie')
wydruk()
```

Przykład dla funkcji z argumentami:

```
def dodawanie(x, y):
    suma = x+y
    return suma                    <- UWAGA!!! return używamy tylko kiedy
                                   oczekujemy zwrotu parametrów z funkcji

def main():
    total = dodawanie(4, 5)
    print(total)

main()
```

Funkcje zagnieżdżone

```
def zewnetrzna_funkcja():
        instrukcje_zewnetrzne
        def wewnetrzna_funkcja():
                instrunkcje_wewnwtrzne
        wewnetrzna_funkcja()
zewnetrzna_funkcja()
```

Funkcje zagnieżdżone pozwalają na:

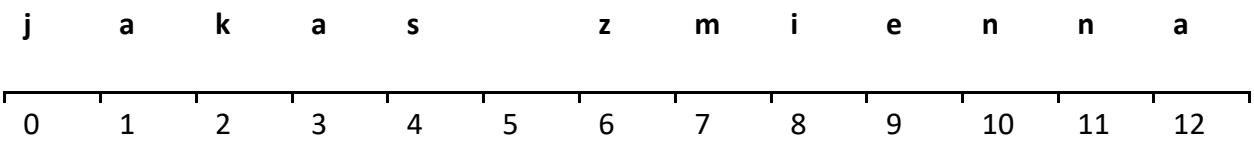
- enkapsulację kodu – użycie pewnej logiki kodu, która odnosi się do konkretnego kontekstu. Pozwala to zachować czystość kodu globalnego. W niektórych przypadkach pozwala to również na ukrycie implementacji kodu
- kod pomocniczy – proste bloki kodu, które są wielokrotnie używane i wspomagają główną logikę programu, ale nie odpowiadają za kluczowe funkcjonalności
- dekoratory – poprawa organizacji i czytelności kodu czyli oddzielenie mniejszych bloków logicznych od większej funkcji, podział bardziej skomplikowanej logiki na mniejsze kroki
- domknięcia funkcji fabrycznych – funkcji które opisują pewne zachowania natomiast nie są zawarte w klasie

Control Flow

W programowaniu ważna jest kolejność wykonywania instrukcji.
Na początku powiedziano, że instrukcje są wykonywane linijka po linijce.
Ale…
Zostały wprowadzone elementy, które powodują obejście niektórych linii lub ich powielanie (if, while, for)
Odnosi się to również do funkcji.
Dlatego też należy znać zasady kontroli sterownia i upewnić się, że instrukcje wykonywane są w odpowiedniej kolejności.

ciągi znaków

- Każda wartość typu string to sekwencja/ciąg znaków, np.: zmienna = ‘jakas zmienna’
- Indeksy ciągów zaczynają się od 0



- indeksy podajemy w nawiasach kwadratowych []
- indeksy zawsze muszą być liczbą całkowitą (int)
- można używać indeksów wstecz (-1, -2…)

Przykład:
zmienna = ‘inna zmienna’
litera = zmienna[4]
print(litera) wynikiem zmiennej o nazwie litera będzie “ ”.

- indeks musi być liczbą mniejszą lub równą ilości znaków w sekwencji
- można zastosować funkcję len(), która zwraca długość stringa
- funkcję len() można zastosować jako parametr określający miejsce w ciągu

Przykład:
zmienna = ‘to jest jakiś ciąg znaków’
ostatnia_litera = zmienna[len(zmienna)-1]
print(ostatnia_litera) wynik: ‘w’

ciągi w pętlach

- można zastosować ciągi znaków zarówno w pętli while I for
- wiele obliczeń polega na przetwarzaniu sekwencji znak po znaku

Przykłady:

Pętla while

```
zmienna=‘kot’
indeks = 0
while indeks < len(zmienna):
    litera = zmienna[indeks]
    print(litera)
    indeks += 1
```

pętla for

```
zmienna=‘kot’
for litera in zmienna:
print(litera)
```

wycinki

- możliwe jest wycięcie określonego kawałka tekstu za jednym razem
- w tym celu do ideksu wstawiamy dwukropek

Przykład:
zmienna = ‘jakas zmienna’
wycinek1 = zmienna[:5] -> ‘jakas’
wycinek2 = zmienna[6:] -> ‘zmienna’
wycinek3 = zmienna[3:8] -> ‘as zm’

- wycinek jest w granicach od n do m-1

stałość stringów

- nie jest możliwe nadpisanie pojedynczego znaku (elementu) w ciągu (obiekcie)
- ciąg znaków jest niezmienny
- możliwe jest:
  - nadpisanie całej zmiennej (zamiana ciągu na inny)
  - utworzenie nowej zmiennej i manipulacje pomiędzy zmiennymi

Przykład:
zmienna = ‘jakas zmienna’
nowa_zmienna = ‘inna’ + zmienna[6:]

operacje na stringach

- można sprawdzić czy dana litera występuje w ciągu znaków
  - ‘f’ **in** ‘jakas zmienna’ -> rozwiązaniem będzie *bool*

Przykład:

```
if 'b' in 'abrakadabra':
    print('W słowie "abrakadabra" znajduje się litera "b".')
```

- można porównać ciągi (np.: porządkowanie alfabetyczne)
  - ‘slowo’ > ’litera’ !!! Litery z początku alfabetu mają mniejszą wartość

Przykład:

```
warzywo = input('Podaj warzywo: ')

if warzywo < 'cebula':
    print('Twoje warzywo będzie przed słowem cebula')
elif warzywo > 'cebula':
    print('Twoje warzywo będzie po słowie cebula')
else:
    print('Twoje warzywo to cebula')
```

- można przyrównać 2 ciągi
    - ‘slowo’ == ‘slowo’

Przykład:

```
warzywo = input('Podaj warzywo: ')

if warzywo == 'cebula':
    print('Tak, chodziło o cebulę!')
else:
    print('Nie, chodziło o inne warzywo')
```

Istnieją jednak pewne ograniczenia:

- Wielkie litery będą miały pierszeństwo przed małymi

operator formatowania

var_int = 42
var_float = 3.8
var_string = ‘chomik’

print(‘Mam %d lat’ %var_int)
print(‘Mój wzrost to %g metra’ %var_float)
print(‘Moje ulubione zwierzę to %s’ %var_string)
print('Przez ostatnie %g lata %s zjadł %d marchewki' %(var_float, var_string, var_int))

Listy

- listy są kolejnym przykładem sekwencji

Przykłady list:
[7, 8, 4, 67]
[‘jakas’, ‘sobie’, ‘lista’]
[‘lista’, 7, 8.4, [9, 78]] <- lista zagnieżdżona
[] <- pusta lista

- listy można przypisać do zmiennych

UWAGA!! W przeciwieństwie do ciągu znaków listy są zmienne

- tak samo jak przy stringach, indeksy liczymy od 0
- można podmieniać pojedyncze elementy list

Przykład:
num_list = [8, 56, 73, 79]
num_list[2] = 23

- indeksy działają tak samo jak przy stringach

- jedną z metod poruszania się po listach jest pętla for
- można odczytać kolejno każdy element listy

Przykład:
```
for element in lista:
    print(element)
```

łączenie wielu funkcji

- wykorzystując różne funkcje można przeprowadzić więcej operacji na elementach listy

Przykład:
```
for x in range(len(lista)):                    !!! len() jako zakres nie trzeba -1
    lista[x] = lista[x] *6
```

listy składane [*list comprehension*]

narzędzie pozwalające na tworzenie nowych list na podstawie innych

może od razu uwzględnić modyfikację

odnosząc się do tego samego przykładu co iteracja przez pętlę for:

```
nowa_lista = [print(x) for x in lista]
```

lub bardziej skomplikowany przykład:

```
nowa_lista = [x**3 for x in lista if x>70]
```

operacje na listach

- łączenie

```
x = [1, 2]
y = [3, 4]
z = x+y          ->    wynik: [1, 2, 3, 4]
```

- powtarzanie

```
x=[1,2]*3        ->    wynik: [1,2, 1,2, 1,2]
```

wycinki list

- podobnie jak przy stringach, można otrzymać wycinki list

```
lista = [1, 't', 56, 'numer', 4.5]
lista[1:3]       ->    ['t', 56]
lista[:2]        ->    [1, 't']
lista[3:]        ->    ['numer', 4.5]
lista[:]         ->    [1, 't', 56, 'numer', 4.5]
```

- za pomocą wycinków można zmienić kilka elementów na raz

Przykład:

```
lista = [1, 't', 56, 'numer', 4.5]
lista[2:4] = ['inne', 'elementy']
lista = [1, 't', 'inne', 'elementy', 4.5]
```

metody

Opis wszystkich metod, które można zastosować do list jest dostępny w dokumentacji Pythona: https://docs.python.org/3/tutorial/datastructures.html

można wywołać spis metod w terminalu, używając funkcji dir():
lista = [1, 'string', False, 4.5]

print(dir(lista))

Kolejną przydatną funkcją jest *help()*, która pokazuje w terminalu jak używać metody:
print(help(lista))

Żeby użyć wybraną metodę należy podać nazwę listy i po kropce nazwę metody z argumentami. Poniżej znajdują się najważniejsze metody i przykłady ich użycia.

Dodawanie elementów

append()
lista.append(element) !!! elementem może być string, int, lista…

Przykład:
```
warzywa = ['pomidor', 'seler', 'marchew']
warzywa.append('ziemniak')
print(warzywa)
```
wynik: ['pomidor', 'seler', 'marchew', 'ziemniak']

Zwracanie indeksu

index()
lista.index(element) !!! tym razem szukamy konkretnego elementu

Przykład:
```
warzywa = ['pomidor', 'seler', 'marchew']
jaki_indeks = warzywa.index('seler')
print(jaki_indeks)
```
wynik: 1

Wstawianie elementów

insert()

lista.insert(element) !!! wstawiamy element do listy (niekoniecznie na końcu)

Przykład:

```
warzywa = ['pomidor', 'seler', 'marchew']
warzywa.insert(1, 'ziemniak')
print(warzywa)
```

!!! najpierw podajemy indeks

wynik: ['pomidor', 'ziemniak', 'seler', 'marchew']

Usuwanie elementów

remove()

lista.remove(element)

Przykład:

```
warzywa = ['pomidor', 'seler', 'marchew']
warzywa.remove('seler')
print(warzywa)
```

wynik: ['pomidor', 'marchew']

Sortowanie elementów

sort()

lista.sort() !!! nie podajemy elementów

Przykład:

```
warzywa = ['pomidor', 'seler', 'marchew']
warzywa.sort()
print(warzywa)
```

wynik: ['marchew', 'pomidor', 'seler']

Krotka [Tuple]

- kolejna ze zbiorów
- krotki są niezmienne
- krotki są zbiorami uporządkowanymi
- pozwalają na duplikaty np.: (1, 2, 1)

krotka = (1, ‘t’, 56, ‘numer’, 4.5) !!nawiasy okrągłe

Tworzenie krotki

- przypisanie na zmienną:

krotka = (1, ‘t’, 56, ‘numer’, 4.5)

- używając konstruktora

krotka = tuple((1, ‘t’, 56, ‘numer’, 4.5))

Krotka z jednym elementem

- Żeby stworzyć krotkę z tylko jednym elementem, należy po nim dostawić przecinek

Przykład:
krotka = (1)

wynik: <class int>

krotka = (1,)

wynik: <class tuple>

Typy danych

int	krotka_int = (1, 2 ,3)
float	krotka_float = (1.6, 2.3 ,3.9)
boolean	krotka_bool = (True, False, False)
string	krotka_string = (‘numer’, ‘jeden’)
mieszane	krotka_mix = (1, ‘t’, 56, ‘numer’, 4.5) Boolean

Operacje na krotkach

- podobnie jak w przypadku listy indeksy zapisujemy w nawiasch kwadratowych
- indeksy zaczynają się od 0

```
krotka = (1, 'numer')
print(krotka[1])
```
wynik: 'numer'

Można stosować również funkcje takie jak:

- type()
- len()
- indeksy ujemne

oraz operacje na krotkach:

- wycinki

```
krotka[:5]
krotka[5:]
krotka[2:5]
krotka[-3:-2]
```

- sprawdzenie czy element istnieje

```
if 'numer' in krotka:
    print('istnieje')
```

zmienianie wartości

- ponieważ krotki są niezmienne, trzeba sobie poradzić inaczej

Przykład:

```
krotka = (1, 2, 3)
lista = list(krotka)
lista[1] = 8
krotka = tuple(lista)
```

dodawanie wartości

- krotka nie posiada metody append, należy zastosować inne metody

1. Konwersja do listy

Przykład:
```
krotka = (1, 2, 3)
lista = list(krotka)
lista.append(4)
krotka = tuple(lista)
```

2. Dodanie kilku krotek

Przykład:
```
krotka1 = (1, 2, 3)
krotka2 = (4,)
krotka1 += krotka2
```

rozpakowywanie krotki

- tworzenie krotki nazywa się jej pakowaniem
- ale można też przeprowadzić operację w drugą stronę

Przykład:
```
warzywa = ('marchew', 'pomidor', 'cebula')

pomaranczowy, czerwony, bialy = warzywa

print(pomaranczowy)   wynik: marchew
print(czerwony)       wynik: pomidor
print(bialy)          wynik: cebula
```

UWAGA!!! Liczba zmiennych musi być taka sama jak liczba wartości w krotce

Asterisk (*)

- Jeżeli liczba wartości w krotce przewyższa liczbę zmiennych, można użyć znaku *
- Może wystąpić tylko jeden asterisk na wyrażenie!!!

Przykład:
```
warzywa = ('marchew', 'pomidor', 'cebula', 'pietruszka', 'kalafior')

(pomaranczowy, czerwony, *bialy) = warzywa

print(pomaranczowy)   wynik: marchew
print(czerwony)       wynik: pomidor
print(bialy)          wynik: ['cebula', 'pietruszka', 'kalafior']
```

krotki w pętlach

krotki = (‘marchew’, ‘pomidor’, ‘cebula’)

```
for i in krotki:
    print(i)
```

```
for i in range (len(krotki):
    print(krotki[i])
```

```
i = 0
while i < len(krotki):
    print(krotki[i])
    i += 1
```

łączenie

```
krotka1 = (1, 2, 3)
krotka2 = (4, 5, 6)
krotka3 = krotka1 + krotka2
print(krotka3)
```

wynik: (1, 2, 3, 4, 5, 6)

powielanie

```
krotka = (‘a’, ‘b’, ‘c’)
powielona = krotka * 2

print(powielona)
```

wynik: (‘a’, ‘b’, ‘c’, ‘a’, ‘b’, ‘c’)

metody.. znowu

- count()

zwraca ile razy dana wartość pojawiła się w krotce

```
krotka = (1, 2, 4.5, True, 1, ‘a’, 6, ‘numer’, 1)
licz = krotka.count(1)
print(licz)
```

wynik: 3

- index()

zwraca indeks, na którym dana wartość pojawiła się w krotce

```
krotka = (1, 2, 4.5, True, 1, ‘a’, 6, ‘numer’, 1)
licz = krotka.index(1)
print(licz)
```

wynik: 0

dostęp do elementów zagnieżdżonych

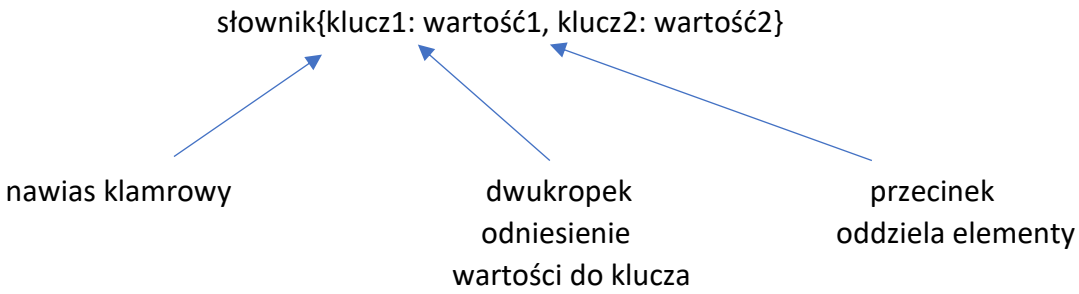
krotka1 = (54, [7, 24, 67], (22, 97, 56))

print(krotka1[2][0])

Słowniki

Tworzenie słowników

Do słownika zawsze dodajemy parę elementów:



funkcja *dict()* tworzy pusty słownik
dict() lub {}

wynik: {}

wyświetlanie słowników

słownik = {1: ‘jeden’, 2: ‘dwa’, 3: ‘trzy’}
print(słownik)

wynik: {1: ‘jeden’, 2: ‘dwa’, 3: ‘trzy’}

print(słownik[1])

wynik: ‘jeden’

poznaną wcześniej funkcję len() można również użyć na słownikach
print(len(słownik))

wynik: 3

typy elementów

- jako klucza można użyć dowolny typ danych (string, int, float, bool..)
- jako wartość można użyć dowolny typ danych
- w jednym słowniku klucze ani wartości nie muszą być tego samego typu

slownik = {1: ‘jeden’, ‘dwa’: 2, 3.5: False}

metody słowników

- tak jak przy stringach I listach można użyć funkcji *dir()* oraz *help()*
- żeby odnieść się do kluczy w słowniku należy użyć metody *keys()*
- żeby odnieść się do wartości w słowniku używamy metody *values()*
- żeby odnieść się do i wartości w słowniku używamy metody *items()*

Przykład:
slownik = {1: ‘jeden’, ‘dwa’: 2, 3.5: False}

```
for i in slownik.keys():
    print(i)
```

```
for i in slownik.values():
    print(i)
```

można użyć tylko nazwę slownika (bez metody), wtedy Python potraktuje to jako metodę *keys():*

```
for i in slownik:
    print(i)
```

```
for i in slownik.keys():
    print(i)
```

dictionary comprehension

Podobnie jak w przypadku list składowych można zastosować dictionary comprehension:

```
slownik = {1:'jeden',
           2:'dwa',
           3:'trzy',
           4:'cztery'}

nowy_slownik = {klucz:wartosc for (klucz,wartosc) in slownik.items()}
print(nowy_slownik)
```

get()

metoda get() pozwala wywołać wartość konkretnego klucza

```
slownik = {1: 'jeden', 'dwa': 2, 3.5: False}
print(slownik.get('dwa'))
```

wynik: 2

```
print(slownik.get(3))
```

wynik: None

sprawdzanie czy klucz jest w słowniku

```
if slownik.get(3) in słownik.values():
    print('Taka wartość jest w słowniku')
else:
    print('Takiej wartości nie ma w słowniku')
```

inna wersja:

```
if 3 in słownik:
    print('Taka wartość jest w słowniku')
else:
    print('Takiej wartości nie ma w słowniku')
```

dodawanie elementów

żeby dodać element do słownika wystarczy odnieść się do jego klucza

```
slownik = {1: 'jeden', 2: 'dwa', 3: 'trzy'}
slownik['nowy'] = 'element'
print(slownik)
```

wynik: {1: 'jeden', 2: 'dwa', 3: 'trzy', 'nowy': 'element'}

Element słownika jest dodawany na końcu nie ma to natomiast znaczenia ponieważ w słownikach nie odnosimy się do indeksu. Nie obchodzi nas czy dany klucz znajduje się w jakiejś konkretnej lokalizacji.

update()

metoda pozwala na aktualizację słownika

slownik = {1: ‘jeden’, ‘dwa’: 2, 3.5: False}
slownik.update({3: ‘trzy’})
print(slownik)

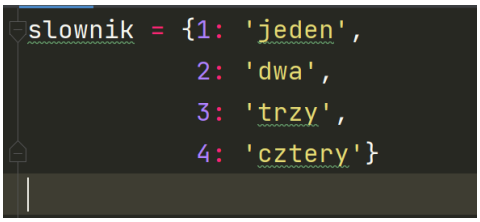
wynik: {1: ‘jeden’, ‘dwa’: 2, 3.5: False, 3: ‘trzy’}

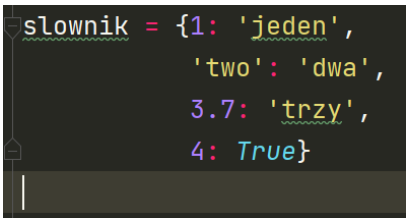
słownik.update(1: ‘niespodzianka’)

wynik: {1: ‘niespodzianka’, ‘dwa’: 2, 3.5: False, 3: ‘trzy’}

przejrzystość kodu

rekomendowane jest definiowanie słownika w kodzie w sposób pokazany poniżej. Pozwala to zachować „czystość kodu” i łatwą identyfikację słownika





Sety

Set jest zbiorem:

- niezmiennym
- nieuporządkowanym
- niezaindeksowanym
- nie pozwala na duplikaty wartości

tworzenie setów

jakis_set = {‘a’, ‘b’, ‘c’} !! nawiasy klamrowe

inną metodą jest set konstruktor:
jakis_set = set((‘a’, ‘b’, ‘c’)) !!! Podwójne nawiasy

właściwości setów

- można używać każdego typu danych
- sety mogą być mieszane
- w mieszanych setach 1 oraz True Python traktuje jako tę samą wartość, podobnie jak w przypadku list, słowników, krotek
- przez sety można się iterować przy pomocy pętli for
- można sprawdzić istnienie wartości w secie

dodawanie elementów

- po utworzeniu setu, nie można go już zmienić, ale można dodawać elementy

- add()

```
jakis_set = {1, 2, 3}
jakis_set.add(4)
```

wynik: {1, 2, 3 ,4}

dodawanie setów

- update()

```
set1 = {1, 2, 3}
set2 = {4, 5, 6}

set1.update(set2)
```

wynik: {1, 2, 3, 4, 5, 6}

dodawanie innych zbiorów

```
set1 = {1, 2, 3}
lista = [4, 5, 6]

set1.update(lista)
```

wynik: {1, 2, 3, 4, 5, 6}

Porównanie zbiorów

Zbiór	zmienny	uporządkowany	duplikaty
lista	**tak**	**tak**	**tak**
krotka	**nie**	**tak**	**tak**
słownik	**tak**	**tak**	**nie**
set	**nie**	**nie**	**nie**

Zastosowania zbiorów

- **Krotka czy lista?** GŁÓWNY WYZNACZNIK: modyfikowalność!!!

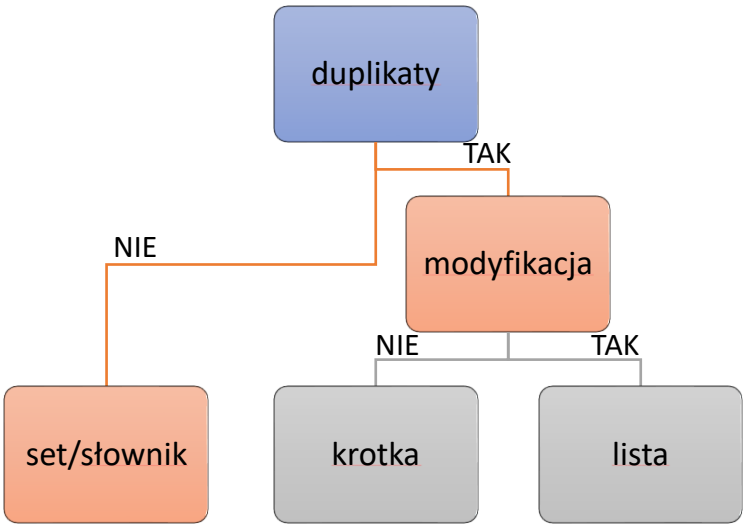
Krotka:
- dane nie powinny lub nie wymagają zmiany
- Iterowanie przez krotki jest szybsze
- Jeśli potrzebujemy tablicę elementów jako klucze słownika, użyj krotki

Lista:
- jeśli potrzebujesz modyfikować zbiór

- **Set czy Lista/Krotka?** GŁÓWNY WYZNACZNIK: duplikaty!!!

Set:
- jeśli potrzebujesz szybko namierzyć czy element jest w zbiorze
- Sety są lepsze jeśli nie potrzebujesz magazynować duplikatów
- Zajmują mniej pamięci i potrzebują mniej czasu operacyjnego

Algorytm wyboru zbioru



Plik

jest sekwencją linii

Podczas podawania nazwy pliku należy podać rozszerzenie

plik = open('jakis_plik.txt', 'r') !!!UWAGA Używamy cudzysłowów przy nazwie pliku i przy modzie

W przypadku niepodania modu zostaną wybrane wartości domyślne (mod1 – odczyt, mod2 – tekst)

Nie podając ścieżki pliku, Python spodziewa się, że plik jest umieszczony w tym samym folderze co skrypt. Jeżeli chcemy odnieść się do innej lokalizacji, należy to uwzględnić podczas otwierania pliku.

plik = open('C:\\Users\PL\Documents\jakis_plik.txt')

funkcja	**mod1**		**mod2**	
open()	r	odczyt	t	tekst
	a	dołączenie	b	binarka
	w	zapis	+	aktualizacja
	x	tworzenie		

Odczyt (mod1 – 'r')

domyślna wartość
Otwiera plik tylko do odczytu.
UWAGA!! Podanie modu, nie powoduje odczytu jego zawartości

Przykład:
plik = open('jakis_plik.txt')
print(plik)
wynik:<_io.TextIOWrapper name='def.txt' mode='r' encoding='cp1252'>

jeżeli plik o podanej nazwie nie istnieje, otrzymujemy **ERROR**

Żeby otrzymać zawartość pliku, należy użyć metody *read()*

Przykład:

```
plik = open('jakis_plik.txt')
print(plik.read())
```

wynik: To jest zawartość tego pliku.

Żeby odczytać część pliku, w nawiasie metody read() należy podać wartość

```
print(plik.read(7))
```

wynik: To jest

Ponieważ plik jest sekwencją linii można na nim zastosować metody jak przy innych sekwencjach w tym iteracje

```
plik = open('jakis_plik.txt')
licznik = 0
for linia in plik:
    licznik += 1
print('Liczba linii w pliku to: ' +str(licznik))
```

Dobrą praktyką jest przechowywanie odczytanej zawartości pliku jako zmiennej

Przykład:

```
plik = open('jakis_plik.txt')
odczyt = plik.read()
```

powinno się zawsze zamknąć plik, po skończonych operacjach

```
plik.close()
```

Zwróccie uwagę, że można użyć innnych metod dla otwartego pliku I dla odczytanego pliku

```
plik = open('jakis_plik.txt')
odczyt =plik.read()
print(dir(plik))
print(dir(odczyt))
```

!!!Tych metod moża użyć do filtrowania tekstu

Dołączanie (mod1 – ‘a’)

- otwiera plik gotowy do modyfikacji
- jeżeli plik nie istnieje, zostanie on stworzony
- zadeklarowana zawartość zostaje dołączona na końcu pliku.

Przykład:

```
plik = open(‘jakis_plik.txt’, ‘a’)
plik.write(‘To jest nowe zdanie’)
plik.close()
```

Zapis (mod1 – ‘w’)

- otwiera plik gotowy do zapisu
- jeżeli plik nie istnieje, zostanie on stworzony
- zadeklarowana zawartość nadpisuje istniejącą zawartość.

Przykład:

```
plik = open(‘jakis_plik.txt’, ‘w’)
plik.write(‘Upps, to nie tak miało być ’)
plik.close()
```

UWAGA!!! operacja otwarcia pliku wystarczy, żeby usunąć istniejącą zawartość

Tworzenie (mod1 – ‘x’)

- nowy, pusty plik jest stworzony
- jeśli plik o takiej nazwie już istnieje, pokaże się **ERROR**

```
plik = open(‘jakis_plik.txt’, ‘x’)
```

readline()

Czyta pojedynczą linię tekstu

Przykład:

```
plik = open('C:\\Users\PL\Desktop\plik.txt')

linia = plik.readline()
linia2 = plik.readline()

print(linia)
print(linia2)

plik.close()
```

readlines()

Czyta wszystkie linie zawarte w pliku, linijka po linijce i zwraca je jako listę stringów. Każdy ze stringów (oprócz ostatniego) zawiera również znak modyfikacji \n.

Szukanie treści w pliku

```
plik = open('plik.txt')
zawartosc = plik.read()
if 'plik' in zawartosc:
   print('ta treść jest w pliku')
else:
   print('Tej treści nie ma w pliku')
zawartosc.index(linia))
```

```
tresc = input('Podaj szukana tresc: ')

plik = open('plik.txt')
zawartosc = plik.readlines()
for linia in zawartosc:
   if linia.find(tresc) != -1:
      print(tresc, 'jest w pliku')
      print('Znajduje się w linijce nr',
      print('ta linijka to:', linia)
```

Pliki do aktualizacji

```
plik = open('jakis_plik.txt', 'r+')
```

r+	odczyt i zapis, dane zostaną nadpisane
w+	zapis i odczyt, dane zostaną nadpisane
a+	dodawanie i odczyt, dane nie zostaną nadpisane

Numpy

NumPy (Numerical Python) jest biblioteką Pythona

Używana do:

- pracy z tablicami
- algebry liniowej
- transformat Fouriera
- macierzy

Open Source Code

Implementacja NumPy

Bibliotekę NumPy, tak jak każdą inną bibliotekę należy zaimportować na początku programu

```
import numpy as np
```

przyjęło się że w bibliotece NumPy nadajemy alias np.

Tablice

Zbiór elementów tego samego typu

NumPy wspiera tablice wielowymiarowe i wielkoformatowe

wbudowany moduł Array	NumPy
import array as arr	import numpy as np
• Potrzeba stosowania dużej ilości pętli • Pozwala na pracę tylko na tablicach jednowymiarowych • Dobre do danych małoformatowych • Obsługuje tylko dane proste dane numeryczne • Nie posiada zaawansowanych funkcji • Dobra wydajność pamięciowa	• Zapewnia narzędzia matematyczne i statystyczne, które mogą być zastosowane do bezpośrednich operacji na tablicach • Operacje wykonywane są na elementach, dzięki czemu nie ma potrzeby stosowania pętli • Pozwala na pracę na tablicach wielowymiarowych • Dobre do danych wielkoformatowych • Obsługuje więcej typów danych • Zintegrowany z innymi bibliotekami • Wydajność pamięciowa pozwala na kontrolę zasobami

Tworzenie tablicy

```
import numpy as np

#tablica jednowymiarowa
tablica_int = np.array([1, 2, 5, 3, 7], dtype=int)
tablica_float = np.array([1, 4.5, 3, 2])
tablica_str = np.array(['dog', '3', 'rower', '3.5'])
tablica_bool = np.array([True, False, True, True], dtype=bool)

#tablica wielowymiarowa
tablica2d = np.array([[1, 2, 8],[5, 3, 7]])
#tablica2d = np.array([[1, 2, 8],
#                      [5, 3, 7]])
```

Wyniki poszczególnych tablic:

```
[1 2 5 3 7]
[1.  4.5 3.  2. ]
['dog' '3' 'rower' '3.5']
[ True False  True  True]
```

```
[[1 2 8]
 [5 3 7]]
```

Wymiar i rozmiar tablicy

Tablica w NumPy może mieć do 32 wymiarów

Wymiar tablicy można sprawdzić używając metody *ndim*

wymiar_tablicy = tablica_int.ndim wynik: 1

Rozmiar tablicy można sprawdzić za pomocą metody *shape*

rozmiar_tablicy = tablica_int.shape wynik: (5,)

Równomierny rozkład wartości

```
# tablica z wartościami z danego zakresu z krokiem
#(początek zakresu, koniec zakresu, krok)
tablica_zakres = np.arange(10)
tablica_zakres_krok = np.arange(1, 10, 2)

# tablica z konkretną ilością wartości
# (początek zakresu, koniec zakresu, ilość punktów)
tablica_ilosc = np.linspace(0, 10, 8)

# reshaping
# (ilość wierszy, ilość kolumn)
zmiana_wymiaru = tablica_ilosc.reshape(4, -1)

# flattening
wyplaszczanie_tablicy = tablica_2d.reshape(-1)
```

Wyniki poszczególnych tablic:

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


[ 0.         1.42857143  2.85714286
4.28571429  5.71428571  7.14285714
 8.57142857 10.        ]
```

W przypadku reshapingu, można wsatwić wartość -1, na którejkolwiek pozycji. W takiej sytuacji NumPy sam dopasuje tę liczbę, aby stworzyć tablicę.

Tablica zer| jedynek

```
# tablica zer
zera = np.zeros(8)
zera2d = np.zeros((5, 4), dtype=int)

# tablica jedynek
jedynki = np.ones(8)
jedynki2d = np.ones((4, 3))

# jedynki na przekątnej
przekatna = np.eye(8)
```

Tablica zer

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

tablica jedynek

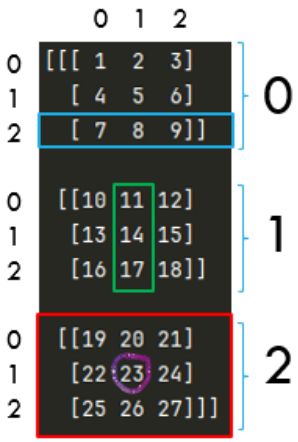
```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

jedynki po przekątnej

```
[[1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1.]]
```

Dostęp do elementów tablicy

Konkretne elementy tablicy można wywołać za pomocą indeksów
tablica[2, 1, 1]
można uzyskać dostęp do całego wiersza, całej kolumny lub całego wymiaru
tablica[0,-1]
tablica[1,:,1]
tablica[2]



Operacje na tablicach

Żeby wykonywać operacje na tablicach, rozmiary tablic muszą się zgadzać

```
tab_1 = np.array([[3, 5, 7, 2],
                  [4, 2, 7, 1]])

tab_2 = np.array([[4, 2, 6, 7],
                  [3, 2, 1, 8]])

tab_3 = np.array([2, 1, 3, 4])

tab_4 = np.array([2, 1])
```

```
# dodawanie/odejmowanie wartości do wszysktich elementów
plus1 = tab_1 + 1
minus3 = tab_2 - 3

# suma wartości tablic
suma = tab_1 + tab_2
inna_suma = np.add(tab_1, tab_3)

tab_1[0] += tab_3
tab_2[:, 0] += tab_4

# suma wszystkich wartości w tablicy
total = tab_1.sum()
```

plus1
[[4 6 8 3]
 [5 3 8 2]]

minus3
[[1 -1 3 4]
 [0 -1 -2 5]]

suma
[[7 7 13 9]
 [7 4 8 9]]

Transponowanie

transponowana = tab_1.T

$$A = \begin{bmatrix} 5 & 3 & -2 & 4 \\ 0 & 10 & 3 & 8 \\ 2 & -1 & 9 & 2 \end{bmatrix} \qquad A^T = \begin{bmatrix} 5 & 0 & 2 \\ 3 & 10 & -1 \\ -2 & 3 & 9 \\ 4 & 8 & 2 \end{bmatrix}$$

Iteracja przez tablicę

Możliwa jest iteracja przez tablicę NumPy
Iteracja odbywa się element po elemencie
W tym celu pomocna jest pętla *for*

```
tablica2d = ([[3, 5, 2], [5, 2, 7]])

# dostęp do elementów tablicy
for element in tablica2d:
    print(element)

#dostęp do poszczególnych wartości tablicy
for element in tablica2d:
    for wartosc in element:
        print(wartosc)
```

=

```
for wartosc in np.nditer(tablica):
    print(wartosc)
```

Łączenie i dzielenie tablic

```
# łączenie tablic
tablica_lacz = np.concatenate((tab_1, tab_2))

# rozdzielanie
tablica_dziel = np.array_split(tab_1, 4, axis=1)
print(tablica_dziel[0], tablica_dziel[1])
czesc1, czesc2, czesc3, czesc4 = tablica_dziel
```

Szukanie wartości w tablicy

Możliwe jest szukanie konkretnej wartości w tablicy
Wynikiem będzie indeks, na którym znajduje się dana wartość (wszystkie przypadki)
W celu wyszukiwania wartości używamy metody *where()*

```
# szukanie wartości
szukaj = np.where(tab_1 == 2)
szukaj_parzyste = np.where(tab_1%2 == 0)
```

(array([0, 1]), array([3, 1]))

Filtrowanie danych

W NumPy filtrowanie elementów odbywa się za pomocą listy bool odnoszących się do numeru indeksu tablicy

```
#filtrowanie
filtr = [True, False, False, True]
nowa_tab = tab3[filtr]
```

Żeby stworzyć filtr dynamiczny używa się instrukcji warunkowych

```
#tworzenie filtra dynamicznego
filtr = []

for element in tab_1:
    if element>5:
        filtr.append(True)
    else:
        filtr.append(False)
```

Maskowanie danych

Metoda celowego ukrywania specyficznych danych w tablic, aby przeprowadzić operacje

```
tablica = np.arange(1,20,2).reshape(2, 5)
maska = tablica>10
zamaskowane = np.ma.masked_where(maska, tablica)


maska2 = np.array([[False, True, False, False, True],[True, True, True, False, True]])
zamaskowane2 = np.ma.masked_array(tablica, maska2)
```

Maski stosuje się:

- kiedy chcemy zachować oryginalne dane, ale nie chcemy kopiować tablicy
- W przypadku, gdy pracujemy na wielu tablicach, żeby uniknąć bugów I skompresować kod
- Gdy dane są brakujące lub nieprawidłowe
- Nie możemy uniknąć operacji na błędnych danych, ale nie chcemy radzić sobie z obiektami typu NaN

Zapisywanie i wczytywanie plików

np.savetxt('plik_testowy.csv', tablica, delimiter=',')

np.load('plik_testowy.csv')

Reprezentacja nieznanych wartości

NaN (Not a Number) – zmiennoprzecinkowe wartości, reprezentujące niezdefiniowane wartości lub wartości, których Python nie jest w stanie rozpoznać

- NaN często reprezentuje brakujące lub błędne dane
- NaN nie mają specyficznej wartości
- Operacje zawierające NaN dają rezultat NaN

None – stała, która sygnalizuje brak wartości

- Używana jako placeholder, jeżeli wartość nie jest podana
- Używana, żeby zasygnalizować brak wartości zwotnej w funkcjach
- Ma swój własny typ: NoneType

N/A lub NA (Not Available) – reprezentuje brakujące, niedostępne lub wartości, których nie można zastosować

- Pliki CSV, Excela, bazy danych mogą zwrócić wartości NA
- Niektóre obiekty w Pythonie traktują NA I NaN naprzemiennie
- Głównie traktowany jako string

Operatory bitowe

&	AND
|	OR
~	NOT

- Operatowy działające na bitach
- Pozwalają na operacje w oparciu o element, zamiast o wartości skalarne
- Używane w logice Pandas i NumPy
- Mają większy priorytet niż “and”, “or” i “not”

ĆWICZENIA

1. Napisz program, który zapyta użytkownika o imię oraz go powita

Przykładowy output:
Jak masz na imię? Paulina
Witaj, Paulina

2. Napisz program, który rozwiąże w jakim wieku są osoby

Maciek ma 21 lat
Ula jest 6 lat starsza niż Maciek
Czarek jest 20 lat starszy niż Ula
Antek ma tyle lat co Czarek i Maciek razem
Kasia ma tyle samo lat co Czarek

Program powinien wydrukować wiek każdej osoby w oddzielnej linii

3. Napisz program, który zapyta użytkownika jakie jest jego ulubione zwierzę, a następnie odpowie, że to też jego ulubione zwierzę

Przykładowy output:
Jakie jest Twoje ulubione zwierzę? **krowa**
Moje ulubione zwierzę to także krowa

4. Napisz program, który zapyta użytkownika o cyfrę, a następnie wydrukuje drugą potęgę tej liczby

Przykładowy output:
Podaj liczbę: **3**
3 do kwadratu to 9

5. Poproś użytkownika o wprowadzenie długości każdego z boków trójkąta, a następnie wydrukuj wartość obwodu

Przykładowy output:
Jaka jest długość pierwszego boku? **1**
Jaka jest długość pierwszego boku? **2**
Jaka jest długość pierwszego boku? **3**
Odwód trójkąta to **6**!

6. Użytkownik powinien wprowadzić temperaturę w stopniach Celsjusza, a program powinien mu powiedzieć jaka to temperaturę w stopniach Farenheita.

Wzór na konwersję:
F=C*9/5+32

Przykładowy output:
Podaj temperaturę [st. C]: **20**
20 stopni Celsjusza to 68 stopni Farenheita

7. Zapytaj użytkownika o podanie 2 liczb, a następnie wyświetl wynik odejmowania drugiej liczby od pierwszej liczby

8. Mad Libs: Napisz program, który zastąpi wyróżnione słowa, używając stałe

Był sobie czarodziej o imieniu **(imię)**, który uwielbiał jeść **(owoc)**.
(imię) zawsze trzymał zapas **(liczba)** **(owoc)** w swojej lodówce!
Pewnego dnia, **(imię)** zdał sobie sprawę, że nie mogą zatrzymać tych wszystkich **(owoc)** dla siebie, więc sprzedał je na targu po **(koszt)** za sztukę,
a za zarobione pieniądze kupił owoce do podzielenia się z całą wioską!
Legenda głosi, że **(ilość_lat)** lat później **(imię)** nadal je owoce.

9. Waga Ziemianina na Marsie wynosi 37,8% ich wagi na Ziemi. Napisz program, który prosi Ziemianina o podanie swojej wagi na Ziemi i wyświetla obliczoną wagę na Marsie.

Przykladowy output:
Podaj wage na Ziemi: **120**
Ekwiwalent na Marsie: 45.36

10. Oblicz pierwiastek kwadratowy liczby podanej przez użytkownika używając biblioteki math

Przykladowy output:
Podaj liczbę: 3
Pierwiastek kwadratowy liczby 3.0 to 1.7320508075688772

11. Napisz program, który symuluje rzut dwiema kostkami i drukuje wyniki każdego rzutu, jak również sumę.

Przykładowy output:
Każda z kości ma 6 ścianek
Pierwsza kość: 3
Druga kość: 5
Suma kości: 8

12. Napisz program, który wygeneruje 2 losowe liczby z zakresu od 0 do 99 i je dodać. Użytkownik próbuje zgadnąć odpowiedź. Program mu mówi czy zgadł dobrze

Oczekiwany output:

Jaka jest suma 58+45?	Jaka jest suma 58+45?
Twoja odpowiedź: 103	Twoja odpowiedź: 130
Odpowiedź prawidłowa!	Źle! Poprawna odpowiedź to 103

13. Napisz program, który zapyta użytkownika o wiek, a następnie powie mu czy może głosować w 3 fikcyjnych krajach: Etgidi (15), Siqira (57), Gmis (25).

Przykładowy output:
Ile masz lat? **21**
Możesz głosować w Etgidi, gdzie wiek wyborczy to 15
Nie możesz głosować w Siqiri, gdzie wiek wyborczy to 57
Nie możesz głosować w Gmis, gdzie wiek wyborczy to 25

14. Napisz program, który zasymuluje rzut dociążoną monetą. Szansa na wylosowanie orła: 70%

15. Napisz program, który powie użytkownikowi czy podany przez niego rok jest przestępny czy nie. Aby rok uznać za przestępny muszą być spełnione warunki:
    - jest podzielny przez 4 i niepodzielny przez 100 lub
    - jest podzielny przez 400

16. Napisz program, który wypisze ciąg Fibbonacciego do określonego limitu (np.: 1000)

17. Napisz program, który pyta użytkownika o liczbę i je dodaje. Program kończy się kiedy użytkownik poda 0.

18. Napisz program, który zasymuluje “magic 8-ball”. Użytkownik zadaje pytanie, na które odpowiedź jest tak lub nie. Magiczna kula pokazuje mu odpowiedź.
    Możliwe odpowiedzi kuli:
    1) Tak
    2) Nie ma mowy
    3) Tylko kula wie
    4) Bezwzględnie
    5) Zapytaj później

    Przykładowy output:
    Czy dostanę awans?
    Tylko kula wie

19. Poproś użytkownika o podanie liczby, następnie napisz program, który policzy od 1 do tej liczby oraz wypisze je. Jeśli liczba jest podzielna przez 3 zastąp ją tekstem “Fizz”, jeśli jest podzielna przez 5 – zastąp ją napisem “Buzz”, natomiast jeśli jest podzielna przez 3 i 5 zastąp ją tekstem “FizzBuzz”. Na koniec podlicz numer “Fizz”, “Buzz” i “FizzBuzz”

Oczekiwany output:
1
2
Fizz
4
Buzz

20. Poproś użytkownika o podanie liczby i napisz program, który doda wszystkie liczby od 1 do podanej (np.: 5 to 1+2+3+4+5)

    Przykładowy output:
    Podaj liczbę: **5**
    Suma to 15

21. Napisz pętlę while, która zaczyna się od ostatniego znaku w napisie i działa od końca, do pierwszego znaku, wypisując każdą literę w osobnej linii, w odwrotnej kolejności.

22. Napisz program, który policzy ile z każdej litery występuje w słowie “rabarbar”

23. Napisz program, który poprosi użytkownika o 5 słów i powie mu które z nich jest najdłuższe oraz ile ma liter

24. Napisz program, który podniesie każdy z elementów listy do kwadratu

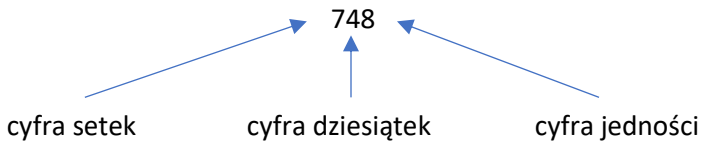
25. Napisz program, który policzy ile jest elementów w liście (bez użycia funkcji len())

26. Napisz program, który doda pola 3 trójkątów

Trójkąt 1:	Trójkąt 2:	Trójkąt 3:
a=3	a=8	a=7
h=5	h=12	h=4

Oczekiwany output:
Pole wszystkich trójkątów to: 69.5

27. Napisz program, który przyjmuje integer i drukuje jego cyfrę jedności



Przykładowy output:
Podaj liczbę: **748**
Cyfra jedności to 8

28. Zamień krotki:

krotka1 = (0, 45, 67)
krotka2 = (99, 65, 32)

Oczekiwany output:
krotka1: (99, 65, 32)
krotka2: (0, 45, 67)

29. Skopiuj kawałek jednej krotki do innej
krotka1 = (54, 7, 24, 67, 22, 97, 56)

Oczekiwany output:
krotka2 = (67, 22, 97)

30. Utwórz słownik, a następnie podnieś jego wszystkie wartości do kwadratu jeśli wartość jest liczbą całkowitą

slownik = {1:1, 2:2.5, 3:3.9, 4:4, 5:4.3,6:4.5}

31. Utwórz krotkę i przerób ją na klucze słownika. Wartości mogą być dowolne.

32. Pamiętacie zadanie z rabarbarem?
Teraz w końcu możecie wybrać dowolne słowo :D
Napisz program, który policzy ile z każdej litery występuje w słowie “rabarbar”

Oczekiwany output:
W slowie "rabarbar" występuja 3 litery "r“
W slowie "rabarbar" występuja 3 litery "a“
W slowie "rabarbar" występuja 2 litery "b"

33. Napisz program, który poda do funkcji listę I wydrukuje kolejno jej elementy
Lista: ['t', [9, 0], 5.7, 6437]

34. Napisz program, który wypisze liczbę elementów, które są unikalne na liście.
lista = [‘k’, 2, 5, 2, ‘mop’, 5.4, ‘k’, 8, False]

Oczekiwany output:
Liczba unikalnych elementów to 7

35. Napisz program, który przyjmie liczbę i wypisze wszystkie możliwe dzielniki tej liczby.

Przykładowy output:
Podaj liczbę: **8**
1
2
4
8

36. Poproś użytkownika, żeby podał string oraz integer. Program powinien wydrukować podany string podaną ilość razy.

Przykładowy output:
Podaj string: **Cześć!**
Podaj liczbę: **3**
Cześć!
Cześć!
Cześć!

37. Zapytaj użytkownika o słowo, następnie jaka to część mowy (rzeczownik, czasownik, przymiotnik)

Jeśli część mowy to 0, zakładamy że słowo to rzeczownik, zdanie brzmi “Super! Od dawna chciałam dodać _____ do mojej kolekcji”
Jeśli część mowy to 1, zakładamy że słowo to czasownik, zdanie brzmi: “Jest taka piękna pogoda, że aż się chce _____”
Jeśli część mowy to 2, zakładamy że słowo to przymiotnik, zdanie brzmi: “Patrząc za okno, niebo jest wielkie I _____”
Weź pod uwagę przypadek, kiedy słowo podane przez użytkownika nie jest ani rzeczownikiem, ani czasownikiem, ani przymiotnikiem

Przykładowy output:
Podaj słowo: **kamień**
Jaka to część mowy? (0 – rzeczownik, 1 – czasownik, 2 – przymiotnik): **0**
Super! Od dawna chciałam dodać kamień do mojej kolekcji

38. Napisz program, który przyjmie liczbę i zwróci jej dwukrotność

Przykładowy output:
Podaj liczbę: **8**
Dwukrotność 8 to 16

39. Napisz program, który powie Ci czy liczba, którą podał użytkownik jest parzysta czy nieparzysta

Przykładowy output:
Podaj liczbę: **43**
43 o liczba nieparzysta

40. Będziemy liczyć do 10, chyba że po drodze nam się odechce
Napisz funkcję, która wydrukuje liczby od 1 do 10 oraz fukncję, która zadecyduje czy chce jej się to liczyć.
Prawdopodobieństwo wydruku każdej z kolejnych liczb to 30%.
Na końcu zawsze musi wyskoczyć napis: “No to tyle”

Przykladowy output:
Będę liczyć do 10, chyba że mi się odechce.
1
2
No to tyle

41. Stwórz pusty plik, następnie poproś użytkownika, żeby podał treść.
Sprawdź czy możesz zapisać tę treść do pliku.
Jeśli tak, to zapisz.
Na koniec powiedz użytkownikowi ile podał znaków.

42. Napisz program, który przeczyta zawartość pliku linijka po linijce in wydrukuje treść w taki sam sposób.

43. Stwórz tablicę, a następnie zamień wszystkie liczby nieparzyste na wartość -1

44. Stwórz tablicę, która będzie posiadała tylko element wspólne poniższych tablic
tab1 = [0 1 2 3 4 5 6 7 8 9]
tab2 = [3 1 13 557 92 0 6 80 43 9]

ODPOWIEDZI:

1.

```
name = input('Jak masz na imię? ')
print('Witaj,', name)
```

2.

```
maciek = 21
ula = maciek + 6
czarek = ula + 20
antek = czarek + maciek
kasia = czarek
print('Maciek ma ', maciek, 'lat')
print('Ula ma ', ula, 'lat')
print('Czarek ma ', czarek, 'lat')
print('Antek ma ', antek, 'lat')
print('Kasia ma ', kasia, 'lat')
```

3.

```
animal = input('Jakie jest Twoje ulubione zwierzę? ')
print('Moje ulubione zwierzę to też', animal)
```

4.

```
num = int(input('Podaj liczbę: '))
pow = num * num
print(num, 'do kwadratu to', pow)
```

5.

```
num1 = int(input('Jaka jest długość pierwszego boku? '))
num2 = int(input('Jaka jest długość drugiego boku? '))
num3 = int(input('Jaka jest długość trzeciego boku? '))
total = num1 + num2 +num3
print('Obwód trójkąta to ' +str(total) +'!')
```

6.

```
celsius = float(input('Podaj temperaturę [st. C] '))
farenheit = celsius * 5/9 +32
print(str(celsius) +' stopni Celsjusza to ' +str(farenheit) +' stopni
Farenheita')
```

7.

```
print('Ten program podaje różnicę.')
num1=float(input('Podaj pierwszą liczbę: '))
num2=float(input('Podaj drugą liczbę: '))
print('Wynik to ' +str(num1-num2))
```

8.

```
IMIE = 'Merlin'
OWOC = 'mango'
LICZBA = 5873
KOSZT = 43
ILOSC_LAT = 200

print('Był sobie czarodziej o imieniu ' +IMIE +', który uwielbiał jeść '
+OWOC +'.')
print(IMIE +' zawsze trzymał zapas ' +str(LICZBA) +' ' +OWOC +' w swojej
lodówce!')
print('Pewngo dnia ' +IMIE +' zdał sobie sprawę, że nie może zatrzymać tych
wszystkich ' +OWOC +' dla siebie,')
print('więc sprzedał je na targu po ' +str(KOSZT) +' za sztukę,')
print('a za zarobione pieniądze kupił owoce do podzielenia się z całą
wioską!')
print('Legenda głosi, że ' +str(ILOSC_LAT) +' lat później ' +IMIE +' nadal je
owoce.')
```

9.

```
MARS_MULTIPLE = 0.378

earth_weight_str = input('Podaj wagę na Ziemi: ')
earth_weight = float(earth_weight_str)
mars_weight = earth_weight * MARS_MULTIPLE
print('Ekwiwalent na Marsie: ' + str(mars_weight))
```

10.

```
import math

num = float(input("Podaj liczbę: "))
root = math.sqrt(num)
print("Pierwiastek kwadratowy liczby ", num, "to", root)
```

11.

```
import random

NUM_SIDES = 6

# random.seed(1)
die1 = random.randint(1, NUM_SIDES)
die2 = random.randint(1, NUM_SIDES)
total = die1 + die2
print("Dice have", NUM_SIDES, "sides each.")
print("First die:", die1)
print("Second die:", die2)
print("Total of two dice:", total)
```

12.

```
import random
MIN_RAND=0
MAX_RAND=99


num1=random.randint(MIN_RAND,MAX_RAND)
num2=random.randint(MIN_RAND,MAX_RAND)
total=num1+num2
print('Jaka jest suma ' +str(num1) +' + ' +str(num2) +'?')
answer=int(input('Twoja odpowiedź: '))
if answer==total:
    print('Odpowiedź prawidłowa!')
else:
    print('Źle! poprawna odpowiedz to ' +str(total))
```

13.

```
ETGIDI=15
SIQIRA=57
GMIS=25

age=int(input('Ile masz lat? '))
if age>ETGIDI:
    print('Możesz głosować w Etgidi, gdzie wiek wyborczy to ' +str(ETGIDI)
+'.')
else:
    print('Nie możesz głosować w Etgidi, gdzie wiek wyborczy to
'+str(ETGIDI) +'.')
if age>SIQIRA:
    print('Możesz głosować w Siqiri, gdzie wiek wyborczy to '+str(SIQIRA)
+'.')
else:
    print('Nie możesz głosować w Siqiri, gdzie wiek wyborczy to
'+str(SIQIRA) +'.')
if age>GMIS:
    print('Możesz głosować w Gmis, gdzie wiek wyborczy to '+str(GMIS) +'.')
else:
    print('Nie możesz głosować w Gmis, gdzie wiek wyborczy to '+str(GMIS)
+'.')
```

14.

```
import random

ORZEL=0.7


if random.random()<ORZEL:
    print("Orzeł")
else:
    print("Reszka")
```

15.

```
year=int(input('Podaj rok. '))
if year%4==0:
    if year%100==0:
        if year%400==0:
            print("To jest rok przestępny")
        else:
            print("To nie jest rok przestępny")
    else:
        print("To jest rok przestępny")
else:
    print("To nie jest rok przestępny")
```

16.

```
LIMIT = 10000
fib1 = 0
fib2 = 1

while fib1 <= LIMIT:
    print(fib1)
    fib_next = fib1 + fib2
    fib1 = fib2
    fib2 = fib_next
```

17.

```
num = int(input("Podaj liczbę: "))
suma = num

while num != 0:
    print("Suma do tej pory to: " + str(suma))
    num = int(input("Podaj liczbę: "))
    suma = suma + num
```

18.

```
import random

ODP_1 = "Tak!"
ODP_2 = "Nie ma mowy."
ODP_3 = "Tylko kula wie."
ODP_4 = "Bezwzględnie."
ODP_5 = "Zapytaj później"

pytanie=input('Zadaj pytanie tak/nie ')

while pytanie!="":

    nr_odp = random.randint(1, 5)
    if nr_odp == 1:
        print(ODP_1)
    if nr_odp == 2:
        print(ODP_2)
    if nr_odp == 3:
        print(ODP_3)
    if nr_odp == 4:
        print(ODP_4)
    if nr_odp == 5:
        print(ODP_5)
    pytanie=input('Zadaj pytanie tak/nie ')
```

19.

```
count_fizz=0
count_buzz=0
count_fizzbuzz=0


num=int(input('Podaj liczbę: '))
for i in range(1,num+1,1):
    if i%5==0 and i%3==0:
        count_fizzbuzz+=1
        print('Fizzbuzz')
    elif i%3==0:
        count_fizz+=1
        print('Fizz')
    elif i%5==0:
        count_buzz+=1
        print('Buzz')
    else:
        print(i)

print()
print('licznik Fizz: ' +str(count_fizz))
print('licznik Buzz: ' +str(count_buzz))
print('licznik FizzBuzz: ' +str(count_fizzbuzz))
```

20.

```
num = int(input('Podaj liczbe: '))
suma=0
for x in range(1, num+1):
    suma+=x
print('Suma to ' +str(suma))
```

21.

```
zmienna = 'jakas zmienna'
indeks=len(zmienna)-1
while indeks >= 0:
    litera = zmienna[indeks]
    print(litera)
    indeks-=1
```

22.

```
slowo = 'rabarbar'
licznik_r = 0
licznik_a = 0
licznik_b = 0

for litera in slowo:
    if litera=='r':
        licznik_r+=1
    elif litera=='a':
        licznik_a+=1
    else:
        licznik_b+=1

print('W slowie "rabarbar" występuja ' +str(licznik_r) +' litery "r"')
print('W slowie "rabarbar" występuja ' +str(licznik_a) +' litery "a"')
print('W slowie "rabarbar" występuja ' +str(licznik_b) +' litery "b"')
```

23.

```
length = 0
for x in range (5):
    word = input('Podaj słowo: ')
    new_len = len(word)

    if new_len>length:
        longest_word = word
        length = len(longest_word)

print('Najdłuższe słowo to ' +longest_word +', a jego długość to ' +str(length))
```

24.

```
list = [4, 6, 2, 8, 1]

for num in range(len(list)):
    squared = list[num]**2
    print(squared)
```

25.

```
list = [7, 'samochod', [3, 6], 'omega', 5.2, 154]
counter = 0

for elemenet in list:
    counter+=1

print('Liczba elementów w liście to: ' +str(counter))
```

26.

```
def pole_trojkata(a, h):
   pole=0.5*a*h
   return pole

def main():
   trojkat_1 = pole_trojkata(3, 5)
   trojkat_2 = pole_trojkata(8, 12)
   trojkat_3 = pole_trojkata(7, 4)

   suma = trojkat_1 + trojkat_2 + trojkat_3

   print(suma)

main()
```

27.

```
def jednosci(num):
   cyfra = num%10
   print(‘Cyfra jedności to 'cyfra)

def main():
   num = int(input('Podaj liczbę: '))
   jednosci(num)

main()
```

28.

```
krotka1 = (0, 45, 67)
krotka2 = (99, 65, 32)

krotka1, krotka2 = krotka2, krotka1

print(krotka1)
print(krotka2)
```

29.

```
krotka1 = (54, 7, 24, 67, 22, 97, 56)
krotka2 = krotka1[3:-1]

print(krotka2)
```

30.

```
slownik = {1:1,
           2:2.5,
           3:3.9,
           4:4,
           5:4.3,
           6:4.5}

nowy_slownik = {klucze:wartosc**2 for (klucze,wartosc) in slownik.items()
if wartosc==int(wartosc)}

print(nowy_slownik)
```

31.

```
jakas_krotka = (4, 3, 'piec', 4.5)
slownik = {}

for element in jakas_krotka:
    slownik[element] = 0

print(slownik)
```

```
jakas_krotka = (4, 3, 'piec', 4.5)
slownik = {klucz:0 for klucz in jakas_krotka}

print(slownik)
```

32.

```
slowo = input("Podaj słowo: ")
slownik = dict()

for element in slowo:
    if element not in slownik:
        slownik[element] = 1
    else:
        slownik[element] += 1

for i in slownik:
    print('W słowie "' +slowo +'" występują ' +str(slownik[i]) +' litery "'
+i +'"')
```

33.

```
def podaj_liste(lista):
    for element in lista:
        print(element)
def main():
    sekwencja = ['t', [9, 0], 5.7, 6437]
    podaj_liste(sekwencja)

main()
```

34.

```
lista = ['k', 2, 5, 2, 'mop', 5.4, 'k', 8, False]
bufor = []
licznik = 0

for element in lista:
    if element not in bufor:
        licznik += 1
        bufor.append(element)

print('Liczba unikalnych elementów to: ', licznik)
```

35.

```
def dzielniki(num):
   for i in range (num):
      obecna_liczba = i+1
      if num%obecna_liczba==0:
         print(obecna_liczba)

def main():
   num = int(input('Podaj liczbę: '))
   dzielniki(num)

main()
```

36.

```
def druk(string, num):
   for i in range (num):
      print(string)

def main():
   string = input('Podaj string: ')
   num = int(input('Podaj liczbę: '))
   druk(string, num)

main()
```

37.

```
def zdanie(slowo, mowa):
    if mowa == 0:
        print('Super! Od dawna chciałam dodać', slowo, 'do mojej kolekcji')
    elif mowa == 1:
        print('Jest taka piękna pogoda, że aż się chce', slowo)
    elif mowa == 2:
        print('Patrząc za okno, niebo jest wielkie i', slowo)
    else:
        print('Podane przez Ciebie słowo nie jest ani rzeczownikiem, ani
czsownikiem, ani przymiotnikiem')

def main():
    slowo = input('Podaj string: ')
    mowa = int(input('Jaka to część mowy? (0 - rzeczownik, 1 - czasownik, 2
- przymiotnik): '))
    zdanie(slowo, mowa)

main()
```

38.

```
def dwukrotnosc(num):
    return num * 2

def main():
    num = int(input('Podaj liczbę: '))
    razy2 = dwukrotnosc(num)
    print('Dwukrotność', num, 'to', razy2)

main()
```

39.

```
def czy_parzysta(num):
    if num % 2 == 0:
        return True

def main():
    num = int(input('Podaj liczbę: '))
    if czy_parzysta(num):
        print(num, 'to liczba parzysta')
    else:
        print(num, 'to liczba nieparzysta')

main()
```

40.

```
import random

PRAWDOPODOBIENSTWO = 0.3

def liczenie():
   for i in range (10):
      obecna_liczba = i+1
      if decyzja():
         return
      print(obecna_liczba)

def decyzja():
   if random.random() < PRAWDOPODOBIENSTWO:
      return True
   return False

def main():
   print('Będę liczyć do 10, chyba że mi się odechce.')
   liczenie()
   print('No to tyle')

main()
```

41.

```
def utworz():
    plik = open('plik.txt', 'x')
    plik.close()

def uzupelnij(tresc):
    plik = open('plik.txt', 'w')
    if plik.writable() == 1:
        plik.write(tresc)
    plik.close()

def przeczytaj():
    plik = open('plik.txt')
    print(plik.read())
    plik.close()

def main():
    utworz()
    tresc = input('Podaj jakis tekst\n')
    uzupelnij(tresc)
    przeczytaj()

main()
```

42.

```
plik = open('plik.txt')
zawartosc = plik.readlines()
for linia in zawartosc:
    print(linia)
```

43.

```
import numpy as np

tab = np.arange(20)
tab[tab%2==1] = -1
print(tab)
```

44.

```
import numpy as np

tab1 = np.arange(10)
tab2 = np.array([3, 1, 13 ,557, 92, 0, 6, 80, 43, 9])
tab3 = np.where(tab1==tab2)
print(tab3)
```