infoShare ACADEMY

Python Podstawy – Intel

infoShare Academy

infoShareAcademy.com

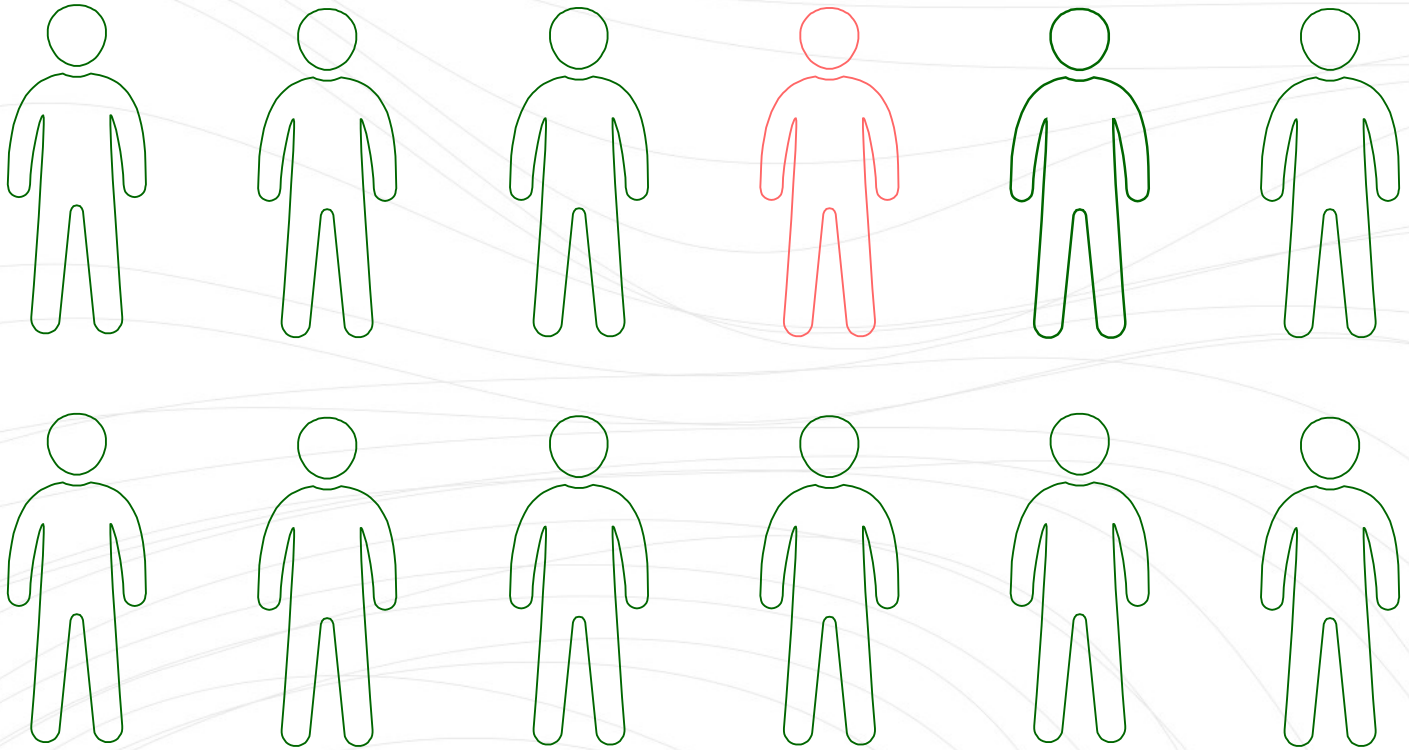
HELLO

Łukasz Falkowicz

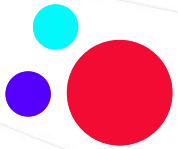
CTO Nais, Trener iSA

infoShare ACADEMY

Poznajmy się ;)

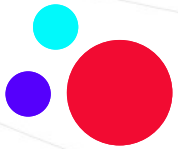


infoShare ACADEMY



Plan na dziś

- Standardy PEP
- Typy danych vol.1
  - int, float, string, bool, None
- Operatory
- Metody wbudowane
- Algorytm
- Bloki kodów
- Instrukcje warunkowe
- Typy danych vol.2
  - range, list, dict, tuple
- Referencja
- Pętle
  - while, for
  - pass, continue, break
  - enumerate, zip



PEP8

Zmienne i Funkcje

snake_case (małe litery, słowa oddzielone podkreśleniami): liczba_klientow, oblicz_sume()

Stałe

CAPS_SNAKE_CASE (duże litery, słowa oddzielone podkreśleniami): MAX_SIZE, PI

Klasy

CamelCase (lub CapWords - słowa zaczynają się z dużej litery, bez spacji): SamochodOsobowy

PEP8

Wcięcia i komentarze

Używaj 4 spacji na wcięcie. Nigdy nie używaj tabulatorów.

Długość Linii

Linie kodu powinny mieć maksymalnie 79 znaków. Jeśli linia jest za długa, należy ją złamać.

Puste Linie

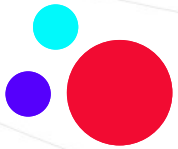
Oddzielaj definicje funkcji i klas dwoma pustymi liniami.

Oddzielaj metody wewnątrz klas jedną pustą linią.

Białe Znaki (Spacje)

Używaj spacji wokół operatorów binarnych (=, +, -, ==, etc.). Przykład: a = b + 1.

Nie używaj spacji bezpośrednio wewnątrz nawiasów, krotek, list czy słowników. Przykład: lista = [1, 2] (nie [1, 2]).



The Zen of Python

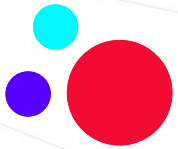
Piękne jest lepsze niż brzydkie.

Wyraźne jest lepsze niż domyślne.

Proste jest lepsze niż złożone.

Złożone jest lepsze niż skomplikowane.

Czytelność się liczy.



Podstawowe typy

1237 – **int** – liczba całkowita

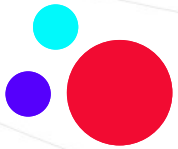
54.3 – **float** – liczba rzeczywista (zmiennoprzecinkowa)

”Ala” – **str** – łańcuch znaków (string)

True / False – **bool** – wartość logiczna prawda / fałsz

None

listy, słowniki, sety, tuple, typy własne...



Zmienne

nazwany obszar pamięci, w którym znajduje się jakaś wartość

pozwala na ponowne użycie wartości w innym miejscu w kodzie

moja_liczba = 1237

nazwisko = ”Kowalski”

czy_obecny = True

Komentarze

#

W Python można zrobić komentarz tylko jednolinijkowy.

Można użyć docstrings do większych komentarzy.

Zazwyczaj używa się potrójnych cudzysłowów ("""..."""").

Operator przypisania

najpierw wykonywane (obliczane) jest wyrażenie, które znajduje się po prawej stronie znaku, następnie ta wartość jest przypisywana do zmiennej po lewej stronie znaku

```
wynik = 2 + 3 ** 3
wynik = 5 != 4 and 'a' not in 'Andrzej'
wynik += 3
```

Operatory

arytmetyczne

+ - * / // % **

porównania (relacyjne)

== != < > <= =>

logiczne

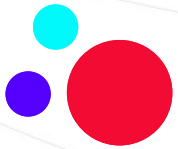
and or not

tożsamości

is is not

członkowstwa

in not in

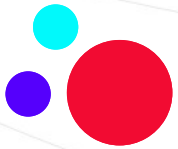


Metody wbudowane

Każdy typ danych posiada zdefiniowane metody (funkcje), które pozwalają na wykonanie różnych (najpopularniejszych) działań.

typ.funkcja()

”ala ma kota”.capitalize()
zdanie = ”ala ma kota”
zdanie.capitalize()



Input / Output

Funkcja **input()** przyjmuje od użytkownika dane i zapisuje do zmiennej.

Wszystko jest stringiem!

```
nazwisko = input(”Podaj nazwisko: ”)
```

Funkcja **print()** służy do wydrukowania tekstu na ekranie.
Automatycznie dodaje na końcu znak specjalny nowej linii **\n**

```
print(nazwisko)
```

String

nazwa_jezyka = ”Python”

len(nazwa_jezyka)
str(jakas_wartosc)

#indeksowanie / wycinanie
nazwa_jezyka[start:koniec:krok]

start - indeks, od którego zaczynamy (włącznie). Domyślnie 0.
koniec - indeks, na którym kończymy (wyłącznie). Domyślnie koniec napisu.
krok - co ile znaków pobierać (np. co drugi). Domyślnie 1.

String – indeksowanie / wycinanie

Znak	P	y	t	h	o	n
Indeks dodatni	0	1	2	3	4	5
Indeks ujemny	-6	-5	-4	-3	-2	-1

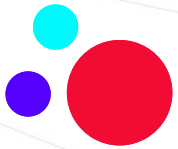
”Python”**[0]**

”Python”**[1:3]**

”Python”**[1:-2]**

”Python”**[::2]**

”Python”**[::-1]**



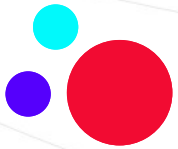
String – formatowanie

```
liczba = 12876.34503
waluta = 'zł'

# F-String
print(f'Wyswietlam liczbę: {liczba} i jeszcze jakiś tekst')
print(f'Wyswietlam wycentrowane: {liczba:-^30}')
print(f'Wyswietlam zaokrąglone: {liczba:.2f}')
print(f'Wyswietlam z separtorem: {liczba:,} {waluta}')
print(f'Wyswietlam z opercją dodatkową: {liczba-1}')

#  format()
print('Wyswietlam: {} i jeszcze jako
        nazwany argument {moja_liczba}'.format(liczba, moja_liczba=liczba))

# modulo (przestarzała metoda)
print('Wyswietlam jako string: %s' % liczba)
print('Wyswietlam jako float: %f %s' % (liczba, waluta))
print('Wyswietlam jako int: %d' % liczba)
```



Int, Float

type(zmienna)

int(zmienna)

float(zmienna)

#int

liczba = 23

#float

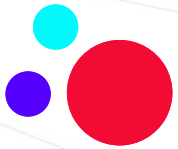
ulamek = 4.53

#sprawdźmy wynik operacji i ich typy

liczba * liczba

ulamek * liczba

liczba + ulamek



Int, Float – operacje

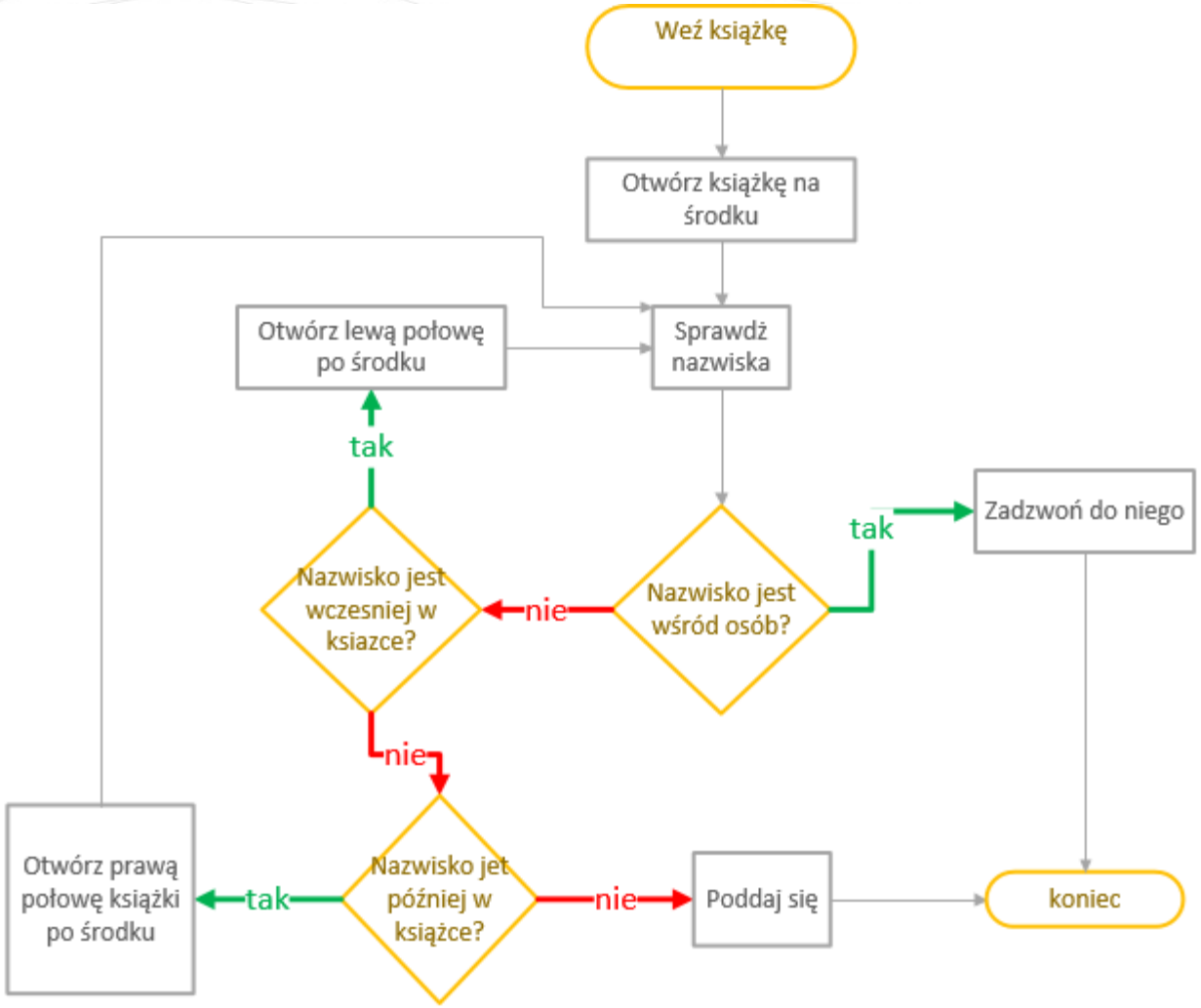
10 + ”a”

*10 * ”-”*

0.1 + 0.2

infoShare ACADEMY

Algorytm

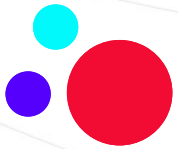


Bloki kodu

indentacja 1 poziom
(wcięcie, 4 spacje)

```
instrukcja/wyrażenie:
    instrukcja
    instrukcja
    instrukcja/wyrażenie:
        instrukcja
instrukcja/wyrażenie:
    instrukcja
```

dwukropek
rozpoczyna blok



Instrukcje warunkowe

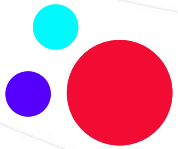
1. weź książkę telefoniczną
2. otwórz książkę na środku
3. sprawdź nazwiska
4. jeśli ”Wojtkowiak” jest wśród osób
5. zadzwoń do niego
6. w przeciwnym razie jeśli ” Wojtkowiak” jest wcześniej w książce
7. otwórz lewą połowę po środku
8. Idź do kroku 3
9. w przeciwnym razie jeśli ” Wojtkowiak” jest później w książce
10. otwórz prawą połowę po środku
11. idź do kroku 3
12. w przeciwnym razie
13. poddaj się

Instrukcje warunkowe

```
if (warunek):
        # jakiś kod wykonany gdy warunek prawdziwy
elif (inny warunek):
        # kod wykonany gdy warunek w if był fałszywy
        # warunek w tym elif musi być prawdziwy aby ten kod wykonać
elif (inny warunek):
        # elif-ów może być wielu. lub żadnego, kod wew. elif wykona się tylko gdy
        # wszystkie wyższe warunki były fałszywe
else:
        # przypadek domyślny, tu nie sprawdzamy warunku, kod w else
        # będzie wykonany gdy wszystkie w if- elif były fałszywe
        # else może być tylko jeden lub wcale
```

Tablica prawdy

A	B	A and B	A or B	not A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True



range()

class range(*stop****)***

class range(*start, stop[, step]****)***

```
# to nie jest funkcja
# sekwencyjny niemutowalny typ danych
# bardzo wydajny
# stop - wyłącznie
```

range(3)

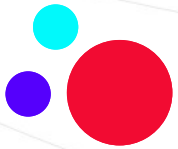
<0, 1, 2>

range(4, 8)

<4, 5, 6, 7>

range(0, 10, 3)

<0, 3, 6, 9>



Listy

class list(*[iterable]****)*** *lub* ***[]***

```
# to nie jest funkcja
# mutowalny typ danych (referencyjny)
# możemy indeksować, slice’ować...
```

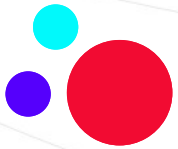
```
lista1 = [1, 2, 3]
lista2 = ["kwiatek", "doniczka", "ziemia", "woda"]
lista3 = []
lista4 = [1, "dwa", 3, 4]
lista5 = list("dwa")
lista6 = list(1)
lista7 = list(range(2,5))
```

Listy zagnieżdżone

```
lista = [ [1,2,3], [4,5,6], [7,8,9] ]
lista = [
            [1,2,3],
            [4,5,6],
            [7,8,9]
        ]

print(lista[1][2])
```

?

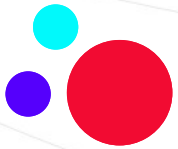


Krotki

class tuple(*[iterable]****)*** *lub* ***()***

```
# to nie jest funkcja
# niemutowalny typ danych
# możemy indeksować, slice’ować...
```

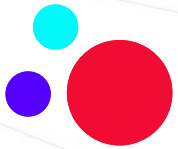
```
krotka1 = (1, 2, 3)
krotka2 = ("kwiatek", "doniczka", "ziemia", "woda")
krotka3 = ()
krotka4 = (1, "dwa", 3, 4)
krotka5 = tuple("dwa")
krotka6 = tuple(1)
krotka7 = tuple(range(2,5))
```



Krotki

```
wyrazy = ("raz", "dwa", "trzy")
wyrazy[0] = "jeden"
print(wyrazy)
```

?



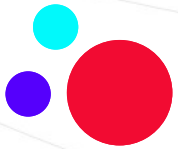
Słowniki

class dict(*[(key1=value1), (key2=value2), ...]****)* *lub* ***{}***

class dict(*key1=value1, key2=value2, ...****)***

```
# to nie jest funkcja
# mutowalny typ danych (referencyjny)
# klucz musi być typem niezmiennym (string, int, tuple)
# klucz jest unikalny w całym słowniku
# wartość może być dowolna i nieunikalna
# odwołujemy się po kluczu
```

```
osoba = {"imie": "Łukasz", "pesel": "123456789"}
print(osoba["imie"])
```



Słowniki

```
osoby = {"studenci": ["Ala", "Jan", "Ania"], "wykladowcy": ["doktor", "profesor"]}
print(osoby["studenci"][1])

osoby["wykladowcy"].append("magister")
osoby["administracja"] = ["pani Basia z dziekanatu"]
osoby.update({"ochrona":"Security Company"})

print(osoby.keys())
print(osoby.values())

for key, item in osoby.items():
        print(key, item)
```

Referencja

Listy, słowniki i zbiory są typami referencyjnymi!

jeśli przypiszemy je innej zmiennej

to tak naprawdę przypiszemy **adres** w pamięci do listy

```
stara_lista = [1,2,3]
nowa_lista = stara_lista
stara_lista[1] = -1
```

w obu zmiennych będzie [-1, 2, 3]

```
nowa_lista = stara_lista.copy()
nowa_lista = list(stara_lista)
nowa_lista = stara_lista[:]

nowa_lista = copy.deepcopy(stara_lista)
```

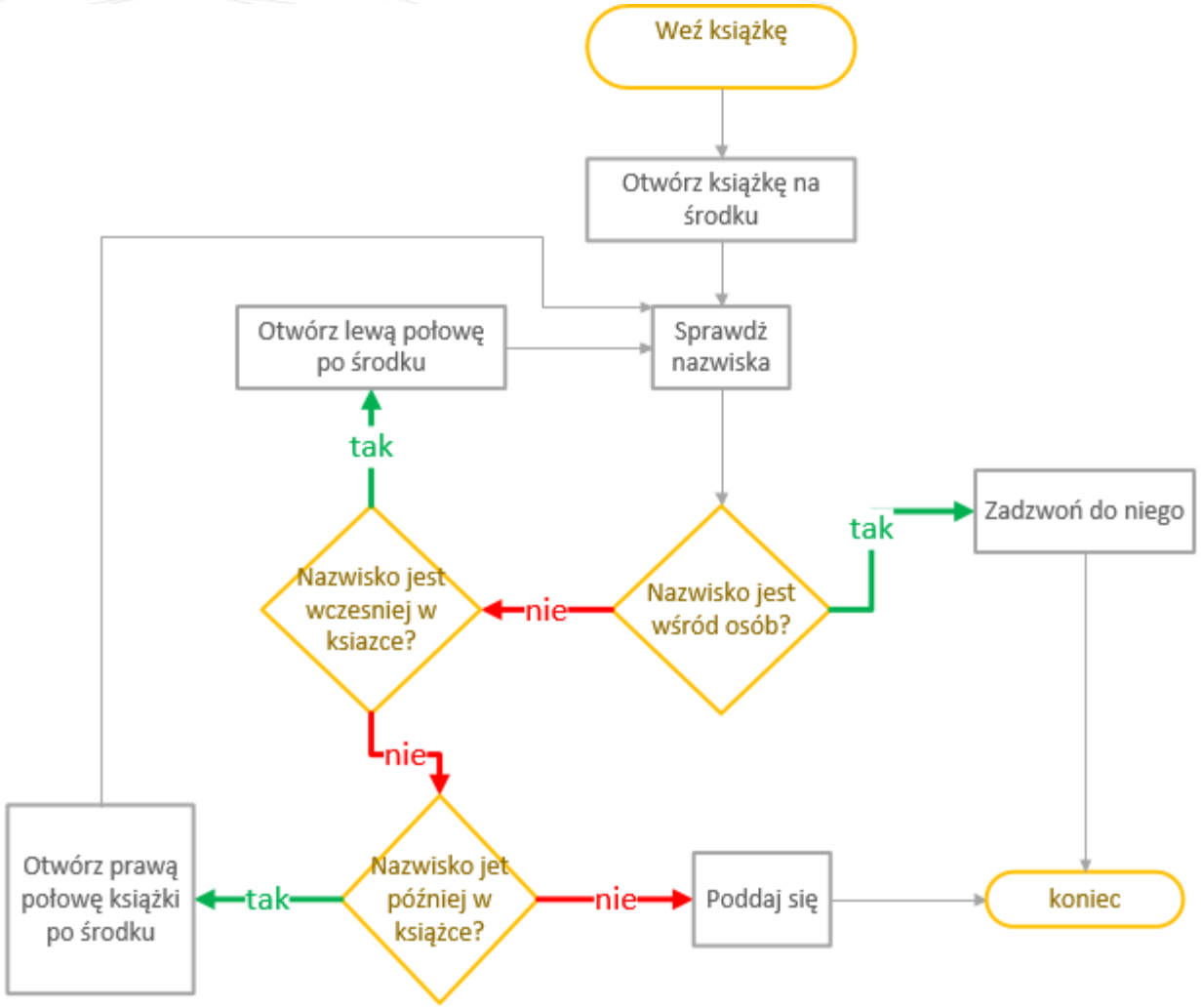
Rozpakowywanie

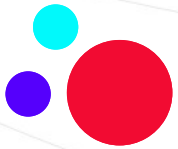
```
cyfry = range(3)
i, j, k = cyfry

liczby = [23, 78, 12]
x, y, z = liczby

wyrazy = ("raz", "dwa", "trzy")
a, b, c = wyrazy
```

Pętle





Pętle

1. weź książkę telefoniczną
2. otwórz książkę na środku
3. sprawdź nazwiska
4. jeśli ”Wojtkowiak” jest wśród osób
5. zadzwoń do niego
6. w przeciwnym razie jeśli ” Wojtkowiak” jest wcześniej w książce
7. otwórz lewą połowę po środku
8. idź do kroku 3
9. w przeciwnym razie jeśli ” Wojtkowiak” jest później w książce
10. otwórz prawą połowę po środku
11. idź do kroku 3
12. w przeciwnym razie
13. poddaj się

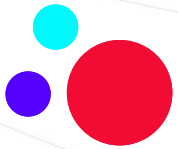
while

```
while (wartość logiczna jest True) :
        # kod który będzie powtarzany tak
        # długo dopóki spełniony będzie warunek
        update wartości logicznej!
```

for

```
for element in zbiór/zakres :
        # kod który wykona się tyle razy ile jest elementów zbioru/zakresu *
        # w tym czasie np. możemy zrobić coś z elementem bo jest on dostępny
        # w ramach pętli for
```

* - są instrukcje którymi możemy to zmienić



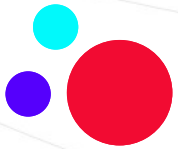
enumerate, zip

for indeks, element **in** ***enumerate(***kolekcja***)*****:**

```
    # enumerate daje nam dwie wartości: indeks
    # bieżącego elementu oraz ten element
```

for element_a, element_b **in** ***zip(****kolekcja_a, kolecja_b****)*****:**

```
    # daje nam elementy z tej samej pozycji w kilku kolekcjach;
    # gdy kolekcje są różnej długości, wielkość najkrótszej
    # kolekcji będzie brana przy ilości powtórzeń pętli
```



pass, continue, break

pass

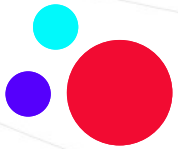
```
# nic nie robi :)
```

continue

```
# program pomija pozostałe instrukcje w bloku i
# wraca do sprawdzenia warunku (while) lub do kolejnego
# elementu/iteracji (for)
```

break

```
# działanie pętli jest przerywane, program przechodzi do
# kolejnej instrukcji po całym bloku pętli
```



for-else, while-else

*Kod wewnątrz bloku **else** wykona się tylko*

*jeśli pętla **NIE** została przerwana instrukcją **break**.*

```
for x in range(0, 10):
    if x == 2:
        print("Istnieje liczba 2 w zbiorze.")
        break
else:
    print("Brak liczby 2 w zbiorze. Nie użyto break wewnątrz pętli.")
```

infoShare ACADEMY

DZIĘKUJĘ NA DZIŚ

Python Podstawy - Intel

infoShareAcademy.com