

Python Instrukcje i pętle



Python Instrukcje i pętle

Agenda

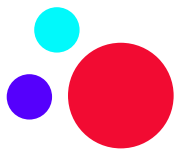
- 1 Instrukcje przypisania
- 2 Instrukcje porównania
- 3 Operatory is/is not
- 4 Operatory bitowe
- 5 Instrukcje warunkowe if-elif-else
- 6 Pętle while/for
- 7 Iteratory range/enumerate



Python Instrukcje i pętle

Agenda

- 1 Instrukcje przypisania
- 2 **Instrukcje porównania**
- 3 Operatory is/is not
- 4 Operatory bitowe
- 5 Instrukcje warunkowe if-elif-else
- 6 Pętle while/for
- 7 Iteratory range/enumerate



Python Instrukcje i pętle

Agenda

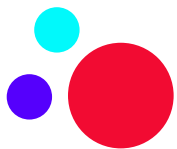
- 1 Instrukcje przypisania
- 2 Instrukcje porównania
- 3 **Operator is/is not**
- 4 Operatory bitowe
- 5 Instrukcje warunkowe if-elif-else
- 6 Pętle while/for
- 7 Iteratory range/enumerate



Python Instrukcje i pętle

Agenda

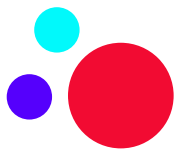
- 1 Instrukcje przypisania
- 2 Instrukcje porównania
- 3 Operatory is/is not
- 4 **Operatory bitowe**
- 5 Instrukcje warunkowe if-elif-else
- 6 Pętle while/for
- 7 Iteratory range/enumerate



Python Instrukcje i pętle

Agenda

- 1 Instrukcje przypisania
- 2 Instrukcje porównania
- 3 Operatory is/is not
- 4 Operatory bitowe
- 5 **Instrukcje warunkowe if-elif-else**
- 6 Pętle while/for
- 7 Iteratory range/enumerate



Python Instrukcje i pętle

Agenda

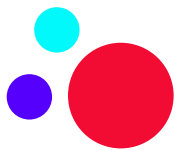
- 1 Instrukcje przypisania
- 2 Instrukcje porównania
- 3 Operatory is/is not
- 4 Operatory bitowe
- 5 Instrukcje warunkowe if-elif-else
- 6 **Pętle while/for**
- 7 Iteratory range/enumerate



Python Instrukcje i pętle

Agenda

- 1 Instrukcje przypisania
- 2 Instrukcje porównania
- 3 Operatory is/is not
- 4 Operatory bitowe
- 5 Instrukcje warunkowe if-elif-else
- 6 Pętle while/for
- 7 **Iteratory range/enumerate**



Python Instrukcje i pętle

Podstawy składni

1. Czytelność kodu.
2. Wcięcia jako element blokowania kodu.
3. Komentarze jako dokumentacja.
4. Zastosowanie nawiasów.



Python Instrukcje i pętle

Podstawy składni

Komentarze są elementami kodu, które nie podlegają wykonaniu. W komentarzu można wpisać dowolną treść i jest ona ignorowana przez interpreter. Zwykle używa się komentarzy do opisu kodu.

W języku Python komentarze oznacza się znakiem krzyżyka #.

komentarz jako linia kodu

```
x = 2 # komentarz za instrukcją
```

```
'''
```

To jest komentarz wieloliniowy w Pythonie.

```
'''
```



Python Instrukcje i pętle

docstring

```
def dodaj(a, b):
```

```
    """
```

```
    Funkcja dodaje dwie liczby.
```

```
    Parameters:
```

```
    a (int): Pierwsza liczba.
```

```
    b (int): Druga liczba.
```

```
    Returns:
```

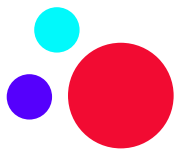
```
    int: Suma dwóch liczb.
```

```
    """
```

```
    return a + b
```

← docstring

info **Share**
ACADEMY



Python Instrukcje i pętle

Komentarze i dobre praktyki

1

Długość komentarza:

Zły przykład:

Funkcja przyjmuje zmienne a, b, c jako input, oblicza sumę zmiennych a + b i zwraca wartość d.

Dobry przykład:

Funkcja zwraca sumę a,b,c

info **Share**
ACADEMY



Python Instrukcje i pętle

Komentarze i dobre praktyki

2

Umiejscowienie komentarza:

Zły przykład:

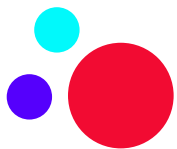
```
return a + b # Funkcja zwraca sumę a,b,c
```

Dobry przykład:

```
# Funkcja zwraca sumę a,b,c
```

```
return a + b
```

info **Share**
ACADEMY



Python Instrukcje i pętle

Instrukcje w Pythonie

W języku Python przyjęto, że jednej instrukcji odpowiada jedna linia kodu. Koniec linii oznacza koniec instrukcji. Linii kodu nie kończy się żadnym innym znakiem.

```
x = 2 #instrukcja1
```

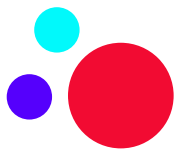
```
y = 3 #instrukcja2
```

```
z = x* y #instrukcja3
```

Możliwe jest również oddzielenie instrukcji przy użyciu średnika np.:

```
x = 2;    y = 3;    z = x * y;
```

lecz nie jest to zalecane.



Python Instrukcje i pętle

Wcięcia vs nawiasy klamrowe

Zasadniczo w językach programowania istnieją dwa podejścia do ograniczania bloków instrukcji.

Są to wcięcia (spacje lub/i tabulatory) i nawiasy.

W języku Python używa się wcięć.

Wcięcia (Python):

```
if x > 0:
```

```
    y = 2 * x
```

Nawiasy (C/C++/C#, Java, JavaScript, R):

```
if (x > 0){
```

```
    y = 2 * x
```

```
}
```



Python Instrukcje i pętle

Wcięcia w Pythonie

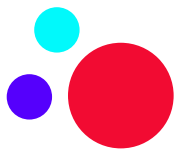
W języku Python struktura kodu jest definiowana poprzez stosowanie wcięć. Stosowanie wcięć wymusza przejrzystość kodu.

Blok kodu jest identyfikowany przez tę samą liczbę wcięć.

```
x = rand() #wybór losowej liczby
w = 3.1415
g = 2 * w

if x > 0:
    y = 2 * x - w
    z = y ** 2*w
    u = z + y * g

print("obliczenia wykonano")
```

Python Instrukcje i pętle

Wcięcia w Pythonie

1. Wcięcia definiują bloki kodu

if warunek:

```
# To jest blok kodu wewnątrz if  
print("Hello, World!")
```

2. Niezmienna liczba spacji lub tabulatorów

Dobre praktyki - użycie spacji (często 4)

if warunek:

```
print("Hello, World!")
```

Złe praktyki - użycie tabulatorów na zmianę ze spacjami

if warunek:

```
print("Hello, World!")  
print ("Hello, Again!")
```



Python Instrukcje i pętle

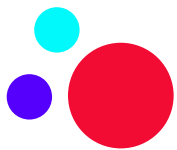
Nawiasy w Pythonie

Nawiasy są elementem języka umożliwiającym grupowanie wyrażeń oraz wymuszania żądanej kolejności wykonania działań.

```
>>> ((2+x)*(y-2))**2
```

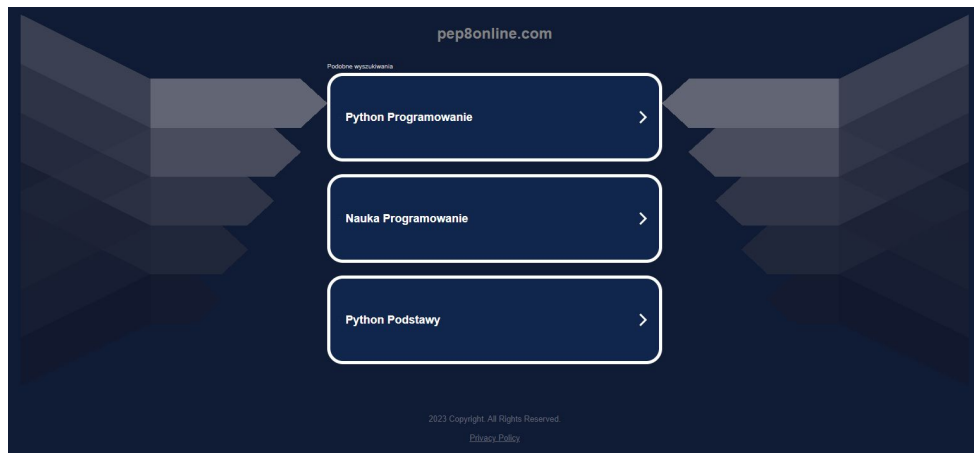
Ponadto nawiasy umożliwiają tworzenie długich wyrażeń przekraczających długość jednej linii np.:

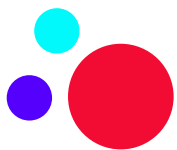
```
>>> ( (2+x) *           # linia 1
... (z+1) -             # linia 2
... (y-2) )             # linia 3
```



Python Instrukcje i pętle

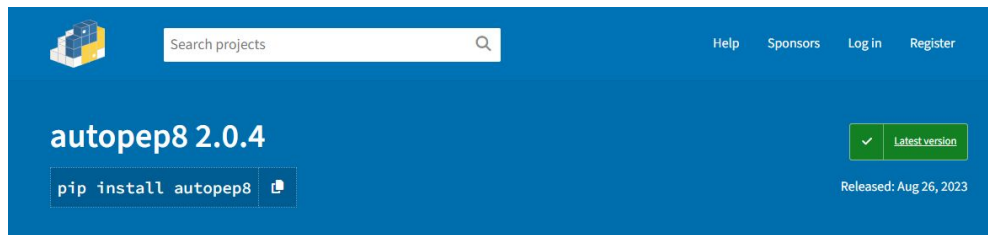
Narzędzia wspomagające





Python Instrukcje i pętle

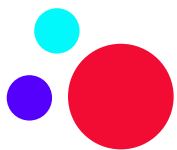
Narzędzia wspomagające



Instalacja i uruchomienie:

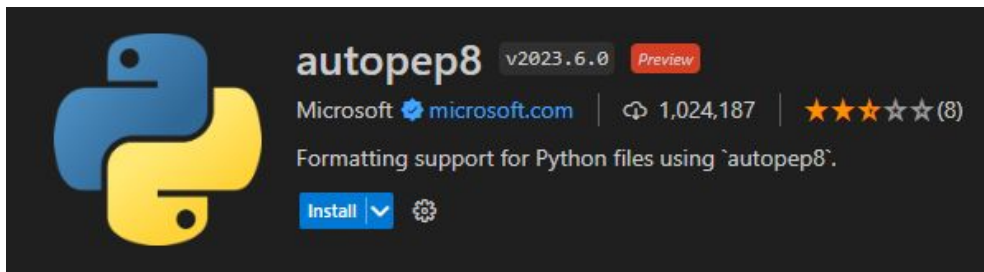
```
>>> pip install autopep8
```

```
>>> autopep8 -in-place nasz_program.py
```

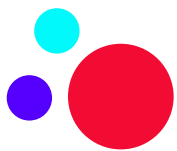


Python Instrukcje i pętle

Narzędzia wspomagające w VSC



info **Share**
ACADEMY



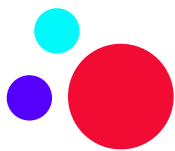
Python Instrukcje i pętle

Narzędzia wspomagające w VSC

Python Docstring Generator

Ctrl + Alt + D

```
"""  
doc  
"""  
autoDocstring - Python Docstring Generator v0.6.1  
Nils Werner | 6,949,372 | ★★★★★ (60)  
Generates python docstrings automatically  
Build Status sponsor  
autoDocstring: VSCode Python Docstring Generator  
Visual Studio Code extension to quickly generate docstrings for python functions.  
test_python.py  
1  
2  
3  
4 def function(number, kwargs=3):  
5     """this is a fun  
6  
7     Arguments:  
8         number ([type]) -- [description]  
9  
10    Keyword Arguments:  
11        kwargs ([type]) -- [description] (default: 3)  
12  
13    Raises:  
14        TypeError -- [description]  
15  
16    Returns:  
17        [type] -- [description]  
18    """  
19  
20    if number < 2:  
21        raise TypeError  
22    return kwargs  
23  
24  
25  
master Python 3.6.2 (3.6.2) Ln 5, Col 21 Spaces: 4 UTF-8 LF Python
```



Python Instrukcje i pętle

Narzędzia wspomagające w VSC

Better Comments

Better Comments v3.0.2
Aaron Bond aaronbond.co.uk | 5,730,328 | ★★★★★ (184) | Sponsor

Improve your code commenting by annotating with alert, informational, TODOs, and more!

[Install](#)

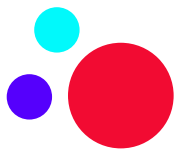
[DETAILS](#) [FEATURE CONTRIBUTIONS](#) [CHANGELOG](#)

Better Comments

The Better Comments extension will help you create more human-friendly comments in your code. With this extension, you will be able to categorise your annotations into:

- Alerts
- Queries
- TODOs
- Highlights
- Commented out code can also be styled to make it clear the code shouldn't be there
- Any other comment styles you'd like can be specified in the settings.

```
1 export class MyClass {
2
3   /**
4    * Important information is highlighted
5    * @deprecated This method is deprecated
6    * @shouldThisMethodBeExposedInPublicAPI
7    * TODO: refactor this method so that it conforms to the API
8    * @param myParam The parameter for this method
9    */
10  public myMethod(myParam: any): void {
11    let myVar: number = 123;
12
13    /** This is highlighted
14     * if (myVar > 0) {
15     *   throw new SystemError(); // This is an alert
16     * }
17
18    /** This is a query
19     * let x = 1;
20
21    // @ts-ignore @ts-nocheck // comment out code
22
23    /** Create some test cases
24
25  }
```

Python Instrukcje i pętle

VSC – przydatne skróty klawiszowe

CTRL/Command + /: zakomentuj bieżącą linijkę

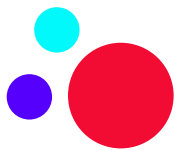
komentarz

CTRL/Command + [: zmień wcięcie linii (w lewo)

```
if True:  
← print("Hello, World!")
```

CTRL/Command +]: zmień wcięcie linii (w prawo)

```
if True:  
→ print("Hello, World!")
```

Python Instrukcje i pętle

Operatory przypisania

>>> x = 5

← wartość zmiennej

↑ nazwa zmiennej

↑ operator przypisania



Python Instrukcje i pętle

Instrukcje przypisania

Przypisania tworzą referencje do obiektów.

Zmienne tworzone są przy pierwszym przypisaniu.

Aby odwołać się do zmiennej konieczne jest uprzednie przypisanie jej wartości.

```
x = 3.1415
```

```
y = ("to", "jest", "krotka", 25)
```

```
a,b = "tekst", 30 # przypisz a="tekst", b=30
```

```
r,p,w = (1,2,3), [5,4], "WWW"
```

```
u = v = d 2.71 # przypisanie wartości 2.71 zmiennym
```



Python Instrukcje i pętle

Przypisanie proste

```
>>> x = 5  
>>> y = 20.5
```

← liczby całkowite i zmiennoprzecinkowe

```
>>> imie = "Ania"
```

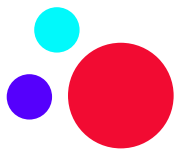
← ciągi znaków

```
>>> liczby = [1, 2, 3]
```

← lista

```
>>> twierdzenie = True
```

← wartości logiczne



Python Instrukcje i pętle

Przypisanie wielokrotne



>>> a, b, c = 1, 2, 3 ← Liczby całkowite lub zmiennoprzecinkowe

>>> imie, nazwisko = "Ania", "Nowak" ← Ciągi znaków

>>> pierwszy, drugi, trzeci = [1, 2, 3] ← Lista

>>> wartosc, walidacja, ilosc = 3.14, True, 100 ← Różne typy danych



Python Instrukcje i pętle

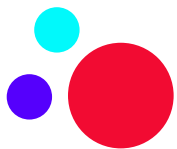
Podstawy składni

>>> x = y = z = 10 ← Liczby całkowite lub zmiennoprzecinkowe

>>> slowo1 = slowo2 = slowo3 = "Python" ← Ciągi znaków

>>> list1 = list2 = list3 = [1, 2, 3] ← Lista

>>> m, n, o = 10, "Hello", [4, 5, 6] ← Różne typy danych



Python Instrukcje i pętle

Przypisanie w miejscu

```
>>> x = 5
```

```
>>> x += 2  
>>> print(x)  
7
```

← równoważne $x = x + 2$

```
>>> x -= 3
```

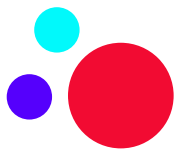
← równoważne $x = x - 3$

```
>>> x *= 2
```

← równoważne $x = x * 2$

```
>>> x /= 4
```

← równoważne $x = x / 4$



Python Instrukcje i pętle

Rozpakowanie sekwencji – listy, krotki

```
>>> lista = [1, 2, 3]
```

```
a, b, c = lista
```

```
print(a) ← wynik: 1
```

```
print(b) ← wynik: 2
```

```
print(c) ← wynik: 3
```

```
>>> krotka = (1, 2, 3)
```

```
a, b, c = krotka
```



Python Instrukcje i pętle

Składnia rozpakowania

Rozpakowanie oznacza automatyczne wyszczególnienie sekwencji i przypisanie jej do zmiennych. Rozpakowanie oznacza się przy pomocy gwiazdki poprzedzającej zmienną.

```
>>> a, *b = [1,2,3,4]
```

```
>>> a
```

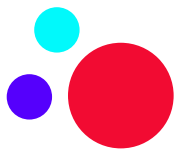
```
1
```

```
>>> b
```

```
[2,3,4]
```

```
>>> wartosci = (1, 2, 3, 4, 5)
```

```
>>> pierwszy, *reszta, ostatni = wartosci
```

Zadanie 6.1 (instrukcja)

1. Utwórz zmienne o nazwach x, y, z i przypisz im wartość 5.
2. Utwórz listę z wartościami [1, 2, 3, 4, 5]. Następnie przypisz pierwszy element do zmiennej first, a resztę do zmiennej rest.
3. Utwórz dwie zmienne, a i b, i przypisz im początkowe wartości. Następnie zamień wartości tych zmiennych, używając jednej linii kodu.
4. Utwórz krotkę z wartościami (10, 20, 30). Przypisz pierwszy element do zmiennej x, drugi do y, a trzeci do z.
5. Utwórz listę z wartościami [1, 2, 3, 4, 5]. Przypisz pierwszy element do zmiennej a, a resztę do zmiennej b przy użyciu gwiazdki (*).
6. Utwórz dwie zmienne a i b, przypisując im wartości 10 i 20. Następnie utwórz zmienną c, przypisując jej sumę a i b.



Python Instrukcje i pętle

boolean

True reprezentuje **prawdziwą** wartość logiczną.

False reprezentuje **falszywą** wartość logiczną.

```
>>>1 == True  
True
```

```
>>>1 is True  
False
```

ALE →

```
>>>0 == False  
True
```

```
>>>0 is False  
False
```

```
>>> True + 1 ← 2
```

```
>>> False + 1 ← 1
```



Python Instrukcje i pętle

Funkcja bool()

```
x = 5  
y = 0
```

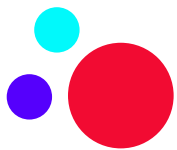
```
bool_x = bool(x)  
bool_y = bool(y)
```

```
print(bool_x)
```

True ← liczba różna od zera jest uznawana za prawdziwą

```
print(bool_y)
```

False ← zero jest uznawane za fałsz



Python Instrukcje i pętle

Kiedy fałsz?

```
>>>bool(None)  
False
```

```
>>>bool(0)  
False
```

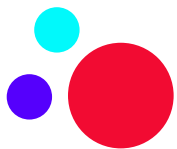
```
>>>bool([])  
False
```



Zadanie 6.2

boolean (instrukcja)

1. Napisz równanie, które zwracać będzie True/False w zależności od tego czy podana liczba całkowita jest parzysta.
2. Napisz równanie, które zwracać będzie True/False przy porównaniu dwóch stringów.
3. Napisz równanie, które zwracać będzie True/False w zależności od tego czy dana lista zawiera przynajmniej jedno zero.
4. Napisz równanie, które zwracać będzie True jeżeli największa liczba jest większa niż średnia wszystkich liczb w liście. W przeciwnym razie niech zwróci False.



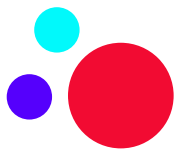
Python Instrukcje i pętle

Określanie warunków

Równe, tożsame, większe, mniejsze.

```
>>> x = 5  
>>> y = 3
```

result = x == y ← False, ponieważ 5 nie jest równe 3.



Python Instrukcje i pętle

Podstawy składni

Wynikiem działania instrukcji porównania są zmienne typu bool. Zmienne te przyjmują dwie możliwe wartości: True oraz False.

```
>>> 2 > 3  
False
```

Możliwe jest tworzenie złożonych warunków składających się z wielu porównań.

```
>>> a < c and c < b
```

 ← tradycyjne podejście

```
>>> a < c < b
```

 ← skrócone podejście



Python Instrukcje i pętle

Operatory porównania

Operatory porównania zwracają wartość **True** jeśli zachodzi dana relacja i wartość **False** w przeciwnym wypadku:

równy ==

nie równy !=

większy >

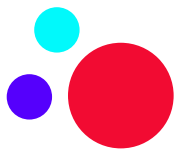
większy

równy >=

mniej <

mniej

równy <=



Python Instrukcje i pętle

Operatory porównania pomiędzy różnymi typami danych

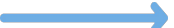
string nie może być konwertowany na **int**

```
>>> 17 == '17'  
False
```

string  int

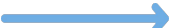
int może być konwertowany do **float**

```
>>> 17 == 17.0  
True
```

int  float

ułamek może być konwertowany do **float**

```
>>> 0.4 == 2/5  
True
```

ułamek  float



Python Instrukcje i pętle

Operatory porównania a NaN

```
>>> x = float('NaN')
```

```
>>> 3 < x
```

False

```
>>> x < 3
```

False

NaN nigdy nie równa się NaN!

```
>>> x == x
```

False



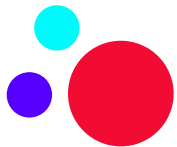
Python Instrukcje i pętle

Operatory porównania – ciągi znaków

Porównanie leksykalne:

```
>>> 'Python' > 'Ruby'  
False
```

```
>>> 'Python' > 'JavaScript'  
True
```



Python Instrukcje i pętle

Operatory porównania – listy, krotki, słowniki

```
>>> [1, 2] == [1, 2]
```

```
True
```

Jeżeli elementy listy są w innej kolejności, w takim wypadku nie są równe:

```
>>> [2, 1] == [1, 2]
```

```
False
```

Tak samo jest w przypadku krotek:

```
>>> (3,4,5) == (5,4,3)
```

```
False
```

Długość listy jest również porównywana:

```
>>> [1, 2] < [1, 2, 3]
```

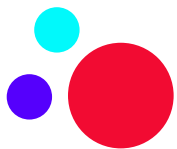
```
True
```



Zadanie 6.3

Operatory porównania (instrukcja)

1. Napisz równanie, które sprawdzi czy suma zmiennych a i b jest mniejsze lub równe c .
2. Napisz równanie, które sprawdzi czy jeden string jest mniejszy od drugiego.
3. Napisz równanie, które sprawdzi czy lista $[1,2,3]$ jest większa od listy $[1,2]$
4. Napisz równanie, które sprawdzi czy słownik $\{ "a": 1, "b": 2 \}$ jest równy słownikowi $\{ "a": 1, "b": 2 \}$
5. Napisz równanie, które sprawdzi czy słownik $\{ "x": 10, "y": 20 \}$ zawiera klucz „z”.



Python Instrukcje i pętle

Operatory logiczne

- **and** zwraca wartość True jeśli oba warunki mają wartość True

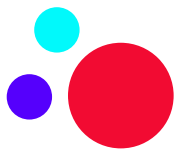
```
>>> x = 5  
>>> y = 10  
>>> (x > 0) and (y < 15) ← True
```

- **or** zwraca wartość True jeśli choć jeden z warunków ma wartość True

```
>>> temperatura = 25  
>>> slonecznie = True  
>>> (temperatura > 30) or slonecznie ← True
```

- **not** jest zaprzeczeniem (jt. zwraca False dla True, oraz True dla False)

```
>>> pada = False  
>>> not pada ← True
```



Python Instrukcje i pętle

Operatory logiczne z bool

```
>>>not True and True  
False
```

```
>>>not (True and False)  
True
```



Python Instrukcje i pętle

Operatory logiczne z bool

a = True
b = False

AND logiczne
True AND False
print(a and b)

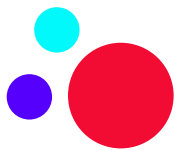
← **False**

OR logiczne
True OR False
print(a and b)

← **True**

NOT logiczne
Not True
print(a and b)

← **False**



Zadanie 6.4

Operatory logiczne (instrukcja)

1. Napisz program, który sprawdza, czy podany rok jest przestępny.
2. Napisz program, który sprawdza, czy podana litera jest samogłoską.
3. Napisz program, który sprawdza, czy podana liczba jest liczbą dwucyfrową.
4. Napisz program, który sprawdza, czy podana liczba jest podzielna zarówno przez 3, jak i przez 5.
5. Napisz program, który sprawdza, czy podana liczba należy do przedziału $(-10, 0)$ lub $(10, 20)$.
6. Napisz program, który sprawdza, czy podana liczba jest dodatnia i nieparzysta.



Python Instrukcje i pętle

Operator **is** oraz **is not**

Instrukcja **is** zwraca wartość prawda jeśli zmienne są referencjami (tj. odnoszą się) do tego samego obiektu.

```
>>> a = 3
>>> b = a
>>> a is b
True
```

Instrukcja **is not** zwraca wartość prawda jeśli zmienne nie są referencjami do tego samego obiektu.

```
>>> a is not b
False
```

Operatorów **is**/**is not** w szczególności używa się do testowania czy zmienna odnosi się do obiektu **None**.



Python Instrukcje i pętle

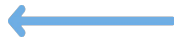
Operator is oraz is not z bool

```
>>>1 == True  
True  
>>>0 == False  
True
```

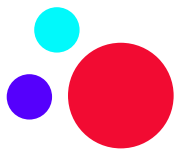


Dla operatora
numerycznego.

```
>>>1 == True  
False  
>>>0 == False  
False
```



Dla operatora
identycznościowego.



Python Instrukcje i pętle

Dobre praktyki

```
>>> flag = True
```

```
>>> if flag == True: ← Nie po „Python’owemu”  
...     print("To porównanie zadziała, ale nie jest uważane za Pythonowe")
```

```
>>> if flag: ← Lepszy sposób  
...     print("Pythonowe rozwiązanie 😊")
```



Python Instrukcje i pętle

Operatory is oraz is not – podsumowanie

1

Porównanie z None:

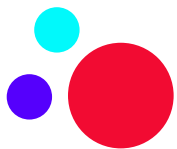
```
>>> x = None
```

```
>>> if x is None:
```

```
    print("Zmienna x jest None.")
```

```
True
```

info **Share**
ACADEMY



Python Instrukcje i pętle

Operatory is oraz is not – podsumowanie

2

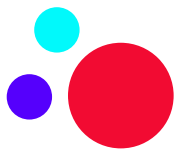
Porównanie obiektów:

```
>>> list1 = [1, 2, 3]
```

```
>>> list2 = list1
```

```
>>> if list1 is list2:
```

```
    print("Obie listy wskazują na ten sam obiekt.")
```



Python Instrukcje i pętle

Operatory is oraz is not – podsumowanie

3

Sprawdzanie Pustego Obiektu:

```
>>> my_list = []
```

```
>>> if my_list is not None and not my_list:
```

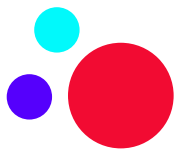
```
    print("Lista jest pusta.")
```



Zadanie 6.5

Operator `is` oraz `is not` (instrukcja)

1. Stwórz dwie listy, gdzie jedna lista będzie kopią drugiej. Następnie sprawdź, czy obie listy wskazują na ten sam obiekt w pamięci, korzystając z operatorów `is` i `is not`.
2. Zdefiniuj zmienną `x` i sprawdź, czy jest ona równa `None`. Użyj operatora `is` w instrukcji warunkowej.
3. Zdefiniuj listę liczb całkowitych i sprawdź, czy jej długość jest równa pewnej liczbie. Użyj operatora `is` w instrukcji warunkowej.



Python Instrukcje i pętle

Operatory bitowe

- $x | y$ bitowa alternatywa x i y
- $x \wedge y$ bitowa różnica symetryczna x i y
- $x \& y$ bitowa koniunkcja x i y
- $x \ll n$ przesunięcie bitowe x w lewo o n bitów
- $x \gg n$ przesunięcie bitowe x w prawo o n bitów
- $\sim x$ uzupełnienie bitowe x



Python Instrukcje i pętle

Operatory bitowe – AND Bitowe (&)

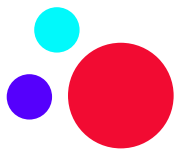
```
>>> x = 5 ← 0b0101  
>>> y = 3 ← 0b0011  
>>> result = x & y  
>>> print(result)  
1 (binarnie: 0b0001)
```



Python Instrukcje i pętle

Operatory bitowe – OR Bitowe (|)

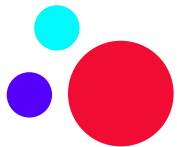
```
>>> x = 5 ← 0b0101  
>>> y = 3 ← 0b0011  
>>> result = x | y  
>>> print(result)  
7 (binarnie: 0b0111)
```



Python Instrukcje i pętle

Operatory bitowe – XOR Bitowe (^)

```
>>> x = 5 ← 0b0101  
>>> y = 3 ← 0b0011  
>>> result = x ^ y  
>>> print(result)  
6 (binarnie: 0b0110)
```

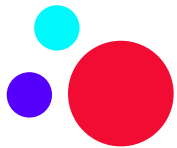


Python Instrukcje i pętle

Operatory bitowe – NOT Bitowe (~)

```
>>> x = 5 ← 0b0101  
>>> result = ~x  
>>> print(result)  
-6 (wynik zależy od liczby bitów w liczbie)
```

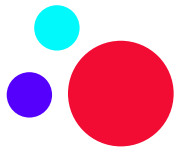
not 0101 = 1010



Python Instrukcje i pętle

Operatory bitowe – Przesunięcie Bitowe w Lewo (<<)

```
>>> x = 5 ← 0b0101  
>>> result = x << 1  
>>> print(result)  
10 (binarnie: 0b1010)
```



Python Instrukcje i pętle

Operatory bitowe – Przesunięcie Bitowe w Prawo (>>)

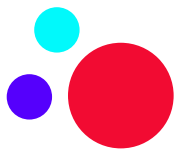
```
>>> x = 5 ← 0b0101  
>>> result = x >> 1  
>>> print(result)  
2 (binarnie: 0b0010)
```



Python Instrukcje i pętle

Operatory bitowe – zastosowanie

1. Manipulacja flagami.
2. Kodowanie binarno-dziesiętne.
3. Optymalizacja.
4. Kryptografia.
5. Obsługa plików binarnych.
6. Algorytmy graficzne.



Python Instrukcje i pętle

Operatory bitowe – podsumowanie

\sim		0	1	$\&$		0	1	$ $		0	1	\wedge		0	1
-----		-----		-----		-----		-----		-----		-----		-----	
		1	0	0		0	0	0		0	1	0		0	1
				1		0	1	1		1	1	1		1	0

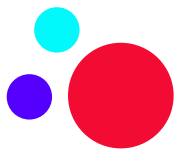
Negacja:
not

Iloczyn:
i

Suma:
lub

Alternatywa:
xor

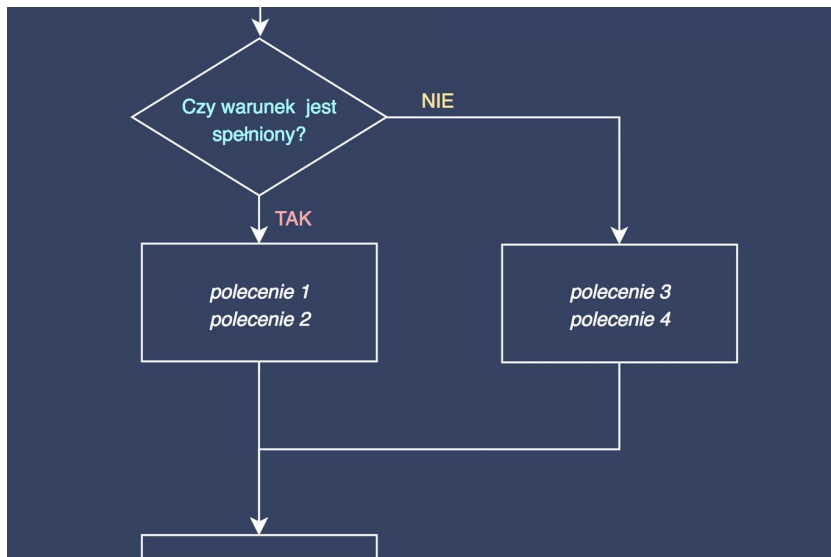
infoShare
ACADEMY



Python Instrukcje i pętle

Instrukcje in/not in oraz warunkowe

info **Share**
ACADEMY





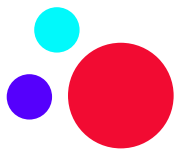
Python Instrukcje i pętle

id obiektu

Identyfikator obiektu:

```
>>> my_list = [1, 2, 3]  
>>> print(id(my_list))
```

2110016417152



Python Instrukcje i pętle

Porównanie id obiektów

```
>>> moje_liczby= [1, 2, 3]
>>> twoje_liczby = moje_liczby
>>> moje_liczby is twoje_liczby
True
```

```
>>> id(moje_liczby)
4517478208
```

← twoje_liczby to alias nazwy
obiektu moje_liczby

Przypisanie nowej nazwy nie powoduje utworzenia nowego obiektu!

```
>>> id(your_fav_numbers)
4517478208
```

```
>>> moje_liczby is not twoje_liczby
False
>>> moje_liczby is not None
True
```

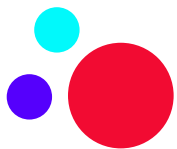


Python Instrukcje i pętle

Porównanie występowania – zbiór

```
>>> moje_liczby= {11, 22, 33}  
>>> 22 in moje_liczby  
True
```

```
>>> 44 in moje_liczby  
False
```



Python Instrukcje i pętle

Porównanie występowania – słownik

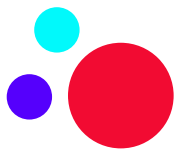
```
>>> pracownicy = {'imie': 'Ania Nowak', 'id': 67826, 'wiek': 33}
```

Czy istnieją klucze w danym słowniku:

```
>>> 'wiek' in pracownicy  
True
```

```
>>> 33 in pracownicy  
False
```

```
>>> 'nazwisko' not in pracownicy  
True
```



Python Instrukcje i pętle

Porównanie występowania – słownik

```
>>> pracownicy = {'imie': 'Ania Nowak', 'id': 67826, 'wiek': 33}
```

```
>>> 33 in pracownicy.values()  
True
```

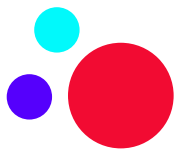


Python Instrukcje i pętle

Porównanie występowania – string

```
>>> kurs = 'Info Share'  
>>> 'Inf' in kurs  
True
```

```
>>> 'Information' in name  
False
```

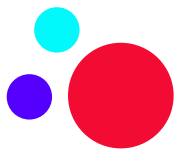



Zadanie 6.6

Operator in (instrukcja)

1. Sprawdź czy element "banana" znajduje się w liście ["orange", "banana", "apple", "blueberry"].
2. Sprawdź czy "hello", znajduje się w słowie "Hello World!".
3. Sprawdź czy klucz "d" NIE znajduje się w słowniku {'a': 1, 'b': 2, 'c': 3}.
4. Sprawdź czy wartość "Gdańsk" istnieje w słowniku:

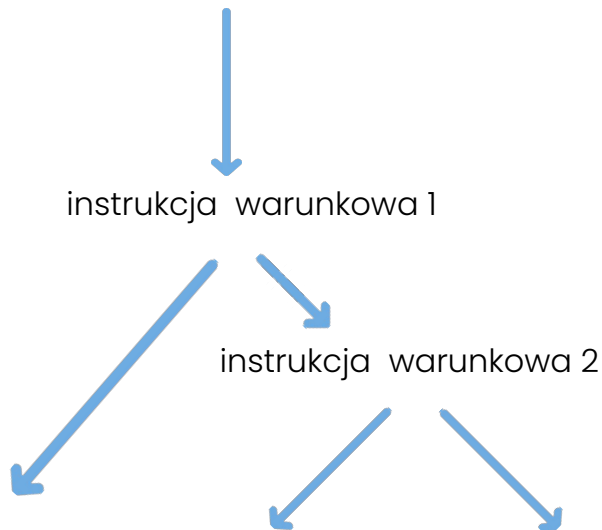
```
dane_osobowe = {  
    "imie": "Jan",  
    "nazwisko": "Kowalski",  
    "wiek": 30,  
    "miasto": "Warszawa",  
    "telefon": "123-456-789"  
}
```



Python Instrukcje i pętle

Instrukcje warunkowe

Instrukcje warunkowe służą do sterowania przebiegiem programu, np.: do podejmowania decyzji w zależności od wartości zmiennej lub zmiennych.



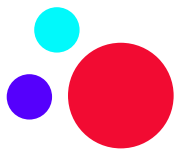


Python Instrukcje i pętle

Instrukcje warunkowe – if

info **Share**
ACADEMY

>>> if condition: ← warunek
wcięcie → # body



Python Instrukcje i pętle

Instrukcje warunkowe – if

info **Share**
ACADEMY

Condition is True

```
number = 10  
if number > 0:  
    # code  
  
# code after if
```

Condition is False

```
number = -5  
if number > 0:  
    # code  
  
# code after if
```



Python Instrukcje i pętle

Instrukcje warunkowe – if-elif

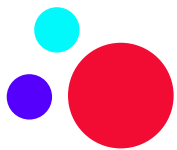
>>> if condition:

—————→ # kod dla condition -> True

else:

—————→ # kod dla condition -> False

info **Share**
ACADEMY



Python Instrukcje i pętle

Instrukcje warunkowe – if-else

info **Share**
ACADEMY

Condition is True

```
number = 10
if number > 0:
    # code

else:
    # code

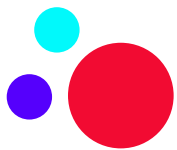
# code after if
```

Condition is False

```
number = -5
if number > 0:
    # code

else:
    # code

# code after if
```



Python Instrukcje i pętle

Instrukcje warunkowe – if-elif-else

if condition1:

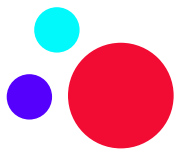
→ # blok 1

elif condition2:

→ # blok 2

else:

→ # blok 3



Python Instrukcje i pętle

if – elif – else

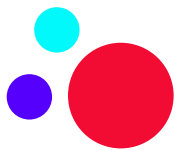
```
if warunek:  
    instrukcje
```

```
if warunek:  
    instrukcje1  
else:  
    instrukcje2
```

```
if warunek1:  
    instrukcje1  
elif warunek2:  
    instrukcje2  
...  
elif warunekN:  
    instrukcjeN
```

```
if warunek1:  
    instrukcje1  
elif warunek2:  
    instrukcje2  
...  
elif warunekN:  
    instrukcjeN  
else:  
    instrukcjeN1
```

info **Share**
ACADEMY



Zadanie 6.7

if-elif-else (instrukcja)

1. Napisz program, który ocenia ocenę ucznia na podstawie uzyskanego wyniku punktowego. Program powinien przyjąć ocenę w skali 0-100, a następnie przypisać ocenę według poniższych kryteriów:
 - Jeśli wynik jest równy lub większy niż 90, przypisz ocenę "A".
 - Jeśli wynik jest równy lub większy niż 80, przypisz ocenę "B".
 - Jeśli wynik jest równy lub większy niż 70, przypisz ocenę "C".
 - Jeśli wynik jest równy lub większy niż 60, przypisz ocenę "D".
 - W przeciwnym razie, przypisz ocenę "F".



Python Instrukcje i pętle

Instrukcja inline if-else

wartość_jeżeli_prawda if warunek else wartość_jeżeli_fałsz

Instrukcje inline if są szczególnie często stosowane w tzw. wyrażeniach składanych.

Np.: przy tworzeniu listy składanej:

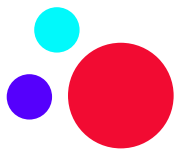
```
>>> [2*i if i>5 else -i for i in range(1,11)] [-1, -2, -3, -4, -5, 12, 14, 16, 18, 20]
```



Zadanie 6.8

inline if-else (instrukcja)

1. Napisz instrukcje inline if-else by móc sprawdzić czy liczba jest parzysta lub nieparzysta.
2. Napisz program, który wczytuje liczbę od użytkownika i wypisuje komunikat "Dodatnia, parzysta i mniejsza niż 20" lub "Inna".
3. Napisz program, który wczytuje tekst od użytkownika i wypisuje komunikat "Dłuższy niż 5 znaków" lub "Krótszy lub równy 5 znakom".
4. Napisz program, który wczytuje dwie liczby od użytkownika i wypisuje komunikat "Są równe" lub "Nie są równe".



Python Instrukcje i pętle

Instrukcja match-case

Instrukcja match-case umożliwia zwięzłe wyrażenie skomplikowanego punktu rozgałęzienia programu. Jest alternatywą dla wielokrotnego użycia if-elif.

```
match subject:
```

```
    case warunek1:
```

```
        instrukcje1
```

```
    case warunek2:
```

```
        instrukcje2
```

```
    case warunek3:
```

```
        instrukcje3
```

```
    case _:
```

```
        instrukcjeN1
```

infoShareAcademy.com

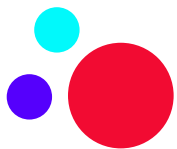
info Share
ACADEMY



Zadanie 6.9

match-case (instrukcja)

1. Stwórz prosty kalkulator, który przyjmuje dwie liczby i operację (dodawanie, odejmowanie, mnożenie, dzielenie). Następnie użyj match-case do obsługi różnych przypadków operacji i zwróć wynik.
2. Użyj match-case do klasyfikacji wieku w kategorii -dziecko (≤ 12 lat) nastolatek (13lat-17lat), dorosłym (18lat-64lat), senior (≥ 65 lat) - i wyświetl odpowiednią wiadomość informującą o kategorii.



Python Instrukcje i pętle

Zagnieżdżanie instrukcji

Instrukcje warunkowe można zagnieżdżać, tzn. umieszczać kolejne instrukcje sterujące w już istniejących. Np.:

```
x,y = 0.5, 0.75
```

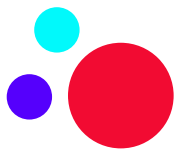
```
a,b,c,d = -1,1,-1,1
```

```
if a<x<b:
```

```
    if c<y<d:
```

```
        print("punkt (x,y) znajduje się wewnątrz prostokąta abcd")
```

info **Share**
ACADEMY



Zadanie 6.10

if zagnieżdżone (instrukcja)

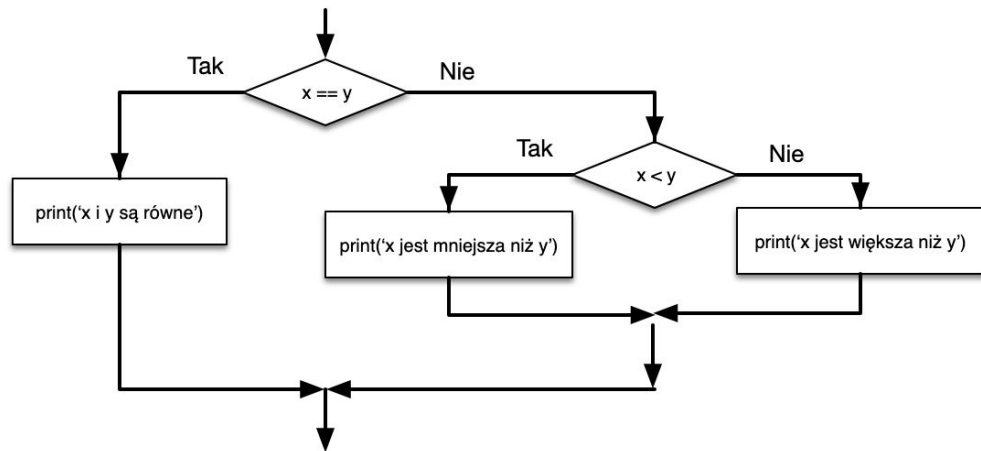
1. Napisz program, który przyjmuje od użytkownika dwie liczby całkowite, a następnie sprawdza, czy:
 - Obie liczby są dodatnie.
 - Jeżeli obie są dodatnie, sprawdź, czy jedna z nich jest parzysta.
 - Jeżeli obie są dodatnie i co najmniej jedna jest parzysta, wyświetl komunikat "Warunki spełnione."
 - Jeżeli któryś z warunków nie jest spełniony, wyświetl odpowiedni komunikat.

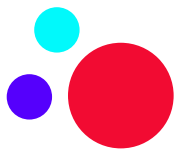


Python Instrukcje i pętle

Instrukcje warunkowe – podsumowanie

info **Share**
ACADEMY





Python Instrukcje i pętle

Pętle

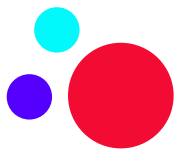
Pętle służą do powtarzania, tj. wielokrotnego wykonania instrukcji.
Powtórzenie zestawu instrukcji w pętli nazywa się iteracją.

Poniższa pętla wykonuje trzy iteracje.

```
print("hello world")  
print("hello world")  
print("hello world")
```

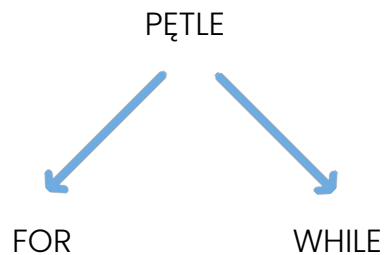


```
for i in range(3):  
    print("hello world")
```



Python Instrukcje i pętle

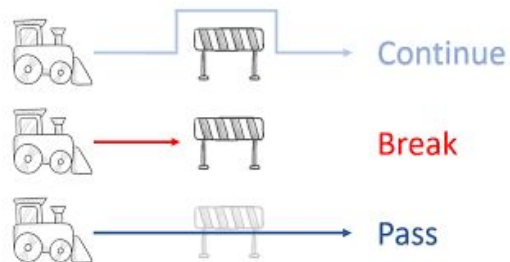
Pętle



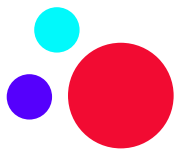


Python Instrukcje i pętle

break, continue, pass



info Share
ACADEMY



Python Instrukcje i pętle

break

```
for i in range(5):  
    if i == 3:  
        break  
    print(i)
```

← Przerzywa wykonywanie pętli.

info **Share**
ACADEMY



Python Instrukcje i pętle

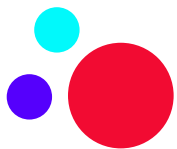
continue

```
for i in range(5):  
    if i == 2:  
        continue  
    print(i)
```



Pomija resztę kodu
w bieżącej iteracji.

info **Share**
ACADEMY

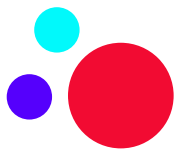


Python Instrukcje i pętle

pass

```
for i in range(5):  
    if i == 2:  
        pass  
    else:  
        print(i)
```

← Nie robi nic.



Python Instrukcje i pętle

Pętla for

Pętla for składa się z licznika, który przyjmuje kolejne wartości z sekwencji lub iteratora oraz powtarzanych instrukcji.

```
>>> for licznik in sekwencja:  
    instrukcje
```



Python Instrukcje i pętle

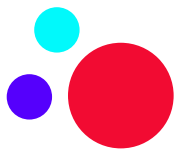
Pętla for – zastosowanie

1

Iteracja po liście:

```
>>> owoce = ["jabłko", "banan", "gruszka"]  
>>> for owoc in owoce:  
    print(owoc)
```

```
jabłko  
banan  
gruszka
```

Python Instrukcje i pętle

Pętla for – zastosowanie

2

Sumowanie liczb w liście:

```
>>> liczby = [1, 2, 3, 4, 5]
>>> suma = 0
>>> for liczba in liczby:
    suma += liczba
>>> print("Suma liczb:", suma)
```

"Suma liczb: 15"

info **Share**
ACADEMY



Python Instrukcje i pętle

Pętla for – zastosowanie

3

Iteracja po ciągu znaków:

```
>>> napis = "Python"  
>>> for litera in napis:  
    print(litera)
```



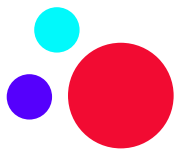
Python Instrukcje i pętle

Pętla for – zastosowanie

4

Iteracja po słowniku:

```
>>> osoba = {"imie": "Anna", "nazwisko": "Kowalska", "wiek": 30}
>>> for klucz, wartosc in osoba.items():
    print(f"{klucz}: {wartosc}")
```



Python Instrukcje i pętle

Pętla for – zastosowanie

5

Iteracja po listach zagnieżdżonych:

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

```
for wiersz in macierz:
```

```
    for element in wiersz:
```

```
        print(element)
```

1
2
3
4
5
6
7
8
9

info **Share**
ACADEMY



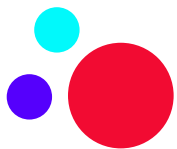
Python Instrukcje i pętle

Pętla for – pułapki

1

Ustalanie warunku zakończenia:

```
for element in sekwencja:  
    if warunek_zakonczenia:  
        break
```



Python Instrukcje i pętle

Pętla for – pułapki

2

Unikanie modyfikacji sekwencji w trakcie iteracji:

```
lista = [1, 2, 3, 4]
```

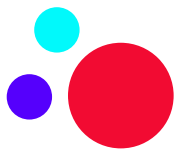
```
for element in lista:
```

```
    lista.append(element * 2)
```



Modyfikacja listy
w trakcie iteracji

info **Share**
ACADEMY



Python Instrukcje i pętle

Pętla for – pułapki

3

Zachowanie zmiennych poza pętlą:

```
for i in range(3):  
    print(i)  
print(i)
```



Potencjalny błąd, jeśli i jest używane poza pętlą

info **Share**
ACADEMY



Zadanie 6.11

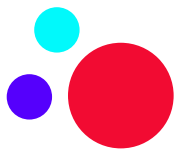
Pętla for (instrukcja)

1. Dana jest lista liczb całkowitych. Napisz pętlę `for`, która stworzy nową listę zawierającą kwadraty liczb z pierwotnej listy.

liczby = [2, 5, 13, 14, 15, 16]

2. Napisz pętlę `for`, która wyświetli wszystkie liczby pierwsze z zakresu od 1 do 20.
3. Dana jest lista. Napisz pętlę `for`, która utworzy nową listę zawierającą elementy w odwrotnej kolejności.

miasta = ["Gdańsk", "Warszawa", "Katowice", "Wrocław", "Poznań"]



Python Instrukcje i pętle

Iteratory

Iteratory są obiektami umożliwiającymi przechodzenie do następnej wartości, ale nie tworzą one obiektu (np. listy czy krotki).

używa sekwencji

```
for i in (0,1,2):  
    print(i)
```

używa iteratora

```
for i in range(3):  
    print(i)
```



Python Instrukcje i pętle

range

Funkcja range zwraca iterator pozwalający licznikowi pętli for przejść od wartości określonej pierwszym argumentem, do wartości określonej drugim argumentem z krokiem podanym jako trzeci argument.

range(wartość_początkowa,wartość_końcowa,krok)

Uwaga: licznik nie przyjmuje wartości argumentu drugiego.

```
>>> for i in range(0,6,2):  
...     print(i) 0  
2  
4
```



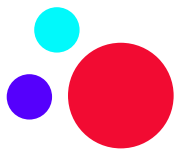
Python Instrukcje i pętle

range

Użycie funkcji `list()` do konwersji obiektu `range` na listę

```
>>> seq = list(range(1, 6))  
>>> print(seq)
```

```
[1, 2, 3, 4, 5]
```



Python Instrukcje i pętle

Funkcja zip

Funkcja zip zestawia ze sobą odpowiadające sobie elementy różnych sekwencji. Funkcja ta zwraca iterator a nie obiekt więc konieczne jest utworzenie na jej podstawie obiektu lub użycie jej jako sekwencji w pętli for.

```
>>> x = (1,2,3)
>>> y = ("a","b","c")
```

```
>>> for i,j in zip(x,y):
...     print((i,j))
```

```
(1, 'a')
(2, 'b')
(3, 'c')
```



Zadanie 6.12

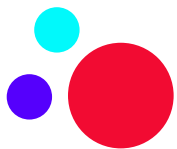
for + range (instrukcja)

1. Mamy dwie listy: uczestnicy zawierającą imiona uczestników oraz punkty zawierającą ilość zdobytych punktów przez każdego z uczestników. Napisz program, który dla każdego uczestnika wyświetli komunikat "Uczeń [imię] zdobył [ilość] punktów.,,

uczestnicy = ['Anna', 'Jan', 'Karolina', 'Piotr']

punkty = [25, 30, 22, 18]

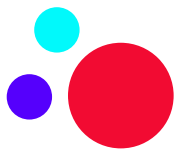
2. Napisz program, który przyjmuje od użytkownika liczbę n i następnie wygeneruje ciąg arytmetyczny, gdzie pierwszy wyraz wynosi 2, a różnica między kolejnymi wyrazami to 3. Wyświetl uzyskany ciąg.



Zadanie 6.13

Podsumowanie (instrukcja)

1. Napisz program, który generuje losowe liczby całkowite z zakresu od 1 do 100 i sprawdza, czy są one parzyste czy nieparzyste. Program powinien następnie zliczać ilość liczb parzystych i nieparzystych oraz wyświetlić wyniki.



Python Instrukcje i pętle-

Podsumowanie



infoShareAcademy.com

info Share
ACADEMY