

Python

Podstawy

Łukasz Kordowski

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy



Historia języka

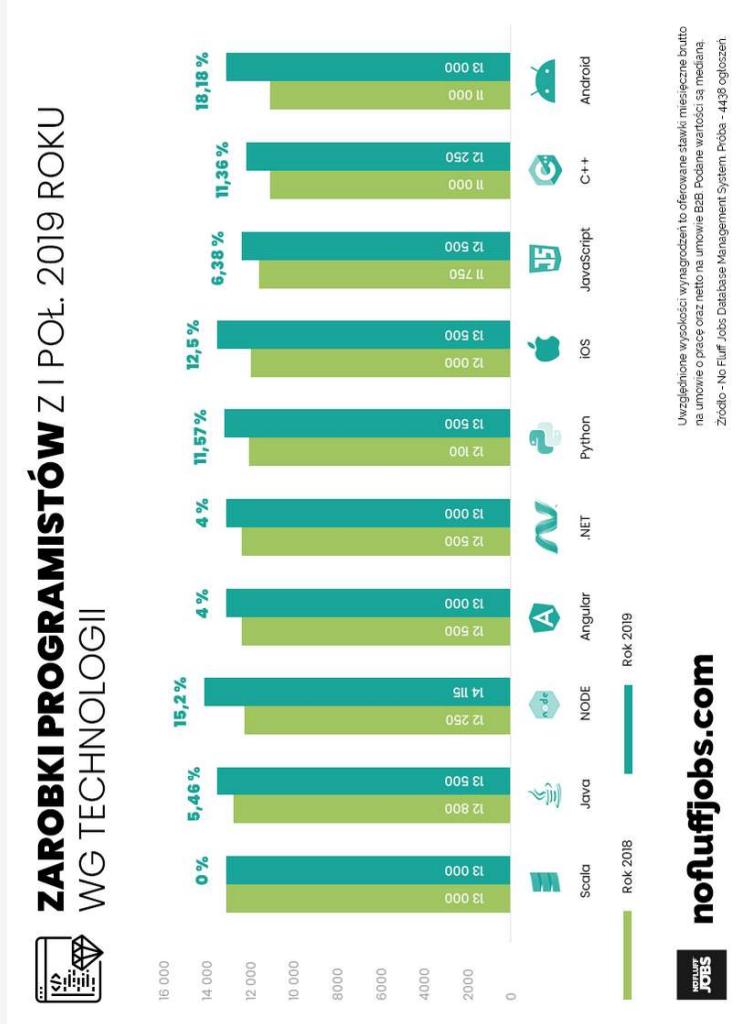


- Napisany przez Guido van Rossum i wydany w 1990 (29 lat temu!) Amsterdam
- Najnowsza stabilna wersja to 3.7.4
- Wersja 2.7 przestanie być wspierana w 2020
- Nazwa nie pochodzi od węża a nazwy programu “Latający cyrk Monty Pythona.”
- Python rozwijany jest jako projekt Open Source zarządzany przez Python Software Foundation



Popularność języka

- Według Stack Overflow [Developer Survey](#) (najbardziej opiniotwórcze badania branży IT) Python jest najszybciej rosnącym w popularności językiem.
- Programiści Pythona to jedni z najlepiej zarabiających na rynku
- Ten kurs to dobry krok w Twojej karierze!



Uwzględnione wysokość wynagrodzeń za dostarczanie stawki miesięcznej brutto na umowę o pracę oraz netto na umowę zatrudnienia. Rodzina wartości sa medianą
źródło - No Fluff Jobs Database Management System. Poda - 4436 ogłoszeń.

nofluffjobs.com

Najważniejsze zastosowania

- Technologie webowe
 - aplikacje webowe (Django, Flask)
 - REST API – interfejsy stron (Django, Flask, aiohttp)
 - Webscraping – indeksowanie stron (requests, BeautifulSoup)
- Obróbka i przetwarzanie danych (pandas, numpy, matplotlib)
 - Praca na plikach tekstowych i arkuszach danych (csv,xlsx)
 - Analiza danych i wizualizacja
- Uczenie maszynowe (Scikit-learn, TensorFlow)
 - Budowanie modeli typujących rozwiązania na podstawie wyuczonych wzorców
 - Sieci neuronowe
- Testy automatyczne (pytest, Selenium)
- Obsługa portów RaspberryPi

i wiele innych

Podstawowe cechy

- **Wysokopoziomowy** – skupiasz się na rozwiązywanym problemie a nie na rejestrach procesora i alokowaniu pamięci
- **Interpretowany** – w przeciwnieństwie do języków komplikowanych, kod wykonywany jest przez dedykowany interpreter
- **Dynamicznie typowany** – typ zmiennej określany jest na podstawie przechowywanej wartości, nie ma potrzeby podawania typu zmiennej przy deklaracji
- Wspiera wiele praw (Paradygmatów) programowania:
 - Obiektowy (klasy, metody, dziedziczenie)
 - Funkcyjny (wyrażenia lambda)
 - Imperatywny (pętle, kolejność wykonania)
 - Refleksyjny (tworzenie obiektów w trakcie działania programu, dostęp do metod)

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Pierwszy program



```
1 print('Hello World')
2
(venv_basic)sda_user@yBox:~/PycharmProjects/python_basics$ python ./tut/hello_world.py
Hello World
```

- Zawartość pliku to zaledwie funkcja wbudowana **print** z przekazanym ciągiem znaków
- Wykonanie skryptu następuje po wywołaniu w terminalu polecenia **python** z podaną ścieżką do skryptu

Filozofia języka



```
1 | import this
```

```
2 |
```

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now,
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

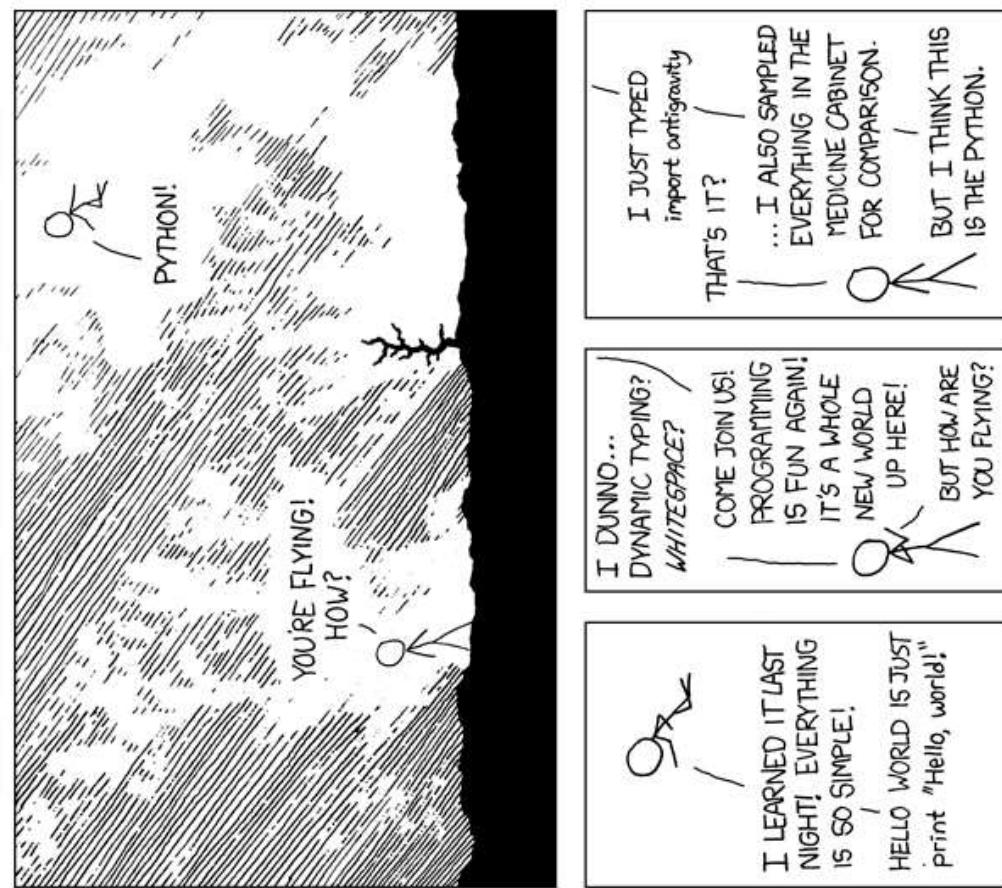
- Filozofia języka Python opisana lirycznie

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Ciekawostka języka

```
1 import antigravity  
2
```



Działanie niektórych wbudowanych modułów może być dość zaskakujące

Autor: Łukasz Kordowski
Prawa do korzystania z materiałów posiada Software Development Academy

Składnia języka



- język C podobny
- formatowanie poziomów programu za pomocą wcięć (unikatowe rozwijazanie), brak znacznika końca linii
- zmienne nie potrzebują deklaracji typu
- snake_case i camelCase

Python

```
1 def silnia(x):  
2     if x == 0:  
3         return 1  
4     else:  
5         return x * silnia(x-1)
```

C

```
1 int silnia(int x) {  
2     if (x == 0) return 1;  
3     else return x * silnia(x-1);  
4 }
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Więcej o składowi języka

- Zbiór dobrych praktyk formatowania kodu definiuje standard [PEP8](#)

Tak

```
x = 1  
y = 2  
long_variable = 3
```

Nie

```
x =  
y =  
long_variable =  
1  
2  
3
```

Cegiełki programowania

- Zmienne
- Pętle
- Funkcje
- Wyrażenia warunkowe
- Operatory
- Struktury danych
- Biblioteka standardowa/wbudowana



Obliczanie BMI

```
1 def calculate_bmi(weight, height):  
2     bmi = weight / height ** 2  
3     if bmi < 18.5:  
4         result = 'underweight'  
5     elif bmi > 25:  
6         result = 'overweight'  
7     else:  
8         result = 'normal'  
9     return result
```

```
Weight in [kg]: 80  
Height in [m]: 1.90  
Your BMI says normal  
  
Wynik  
  
Weight in [kg]: 80  
Height in [m]: 1.90  
Your BMI: 20.41 says normal
```

Wynik z bmi, p06a_bmi.py

```
1 if __name__ == '__main__':  
2     user_weight = float(input('Weight in [kg]: '))  
3     user_height = float(input('Height in [m]: '))  
4     user_result = calculate_bmi(user_weight, user_height)  
5     print(f'Your BMI says {user_result}')
```

p06_bmi.py

Obliczanie BMI - objaśnienie

```
1 def calculate_bmi(weight, height):  
2     bmi = weight / height ** 2  
3     if bmi < 18.5:  
4         result = 'underweight'  
5     elif bmi > 25:  
6         result = 'overweight'  
7     else:  
8         result = 'normal'  
9     return result  
  
10 if __name__ == '__main__':  
11     user_weight = float(input('Weight in [kg]: '))  
12     user_height = float(input('Height in [m]: '))  
13     user_result = calculate_bmi(user_weight, user_height)  
14     print(f'Your BMI says {user_result}')
```

- Definicja funkcji rozpoczyna się znacznikiem **def**
- Funkcje pomagają ograniczyć liczbę linii kodu poprzez swoje wywołanie
- Nazwa funkcji postuguje się konwencją `snake_case`
- W nawiasach okrągłych podajemy argumenty, czyli wartości potrzebne do wykonania funkcji, rozdzielone przecinkiem
- Linia definicji funkcji kończy znak :
- Ciało funkcji znajduje się za wcięciem

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Obliczanie BMI - objaśnienie

```
1 def calculate_bmi(weight, height):  
2     bmi = weight / height ** 2  
3     if bmi < 18.5:  
4         result = 'underweight'  
5     elif bmi > 25:  
6         result = 'overweight'  
7     else:  
8         result = 'normal'  
9     return result  
10  
11 if __name__ == '__main__':  
12     user_weight = float(input('Weight in [kg]: '))  
13     user_height = float(input('Height in [m]: '))  
14     user_result = calculate_bmi(user_weight, user_height)  
15     print(f'Your BMI says {user_result}')  
16  
17
```

- W ciele funkcji wykonujemy potrzebne nam obliczenia
- W tym celu przy pomocy operatora przypisania `=`, zapisujemy wynik obliczeń do zmiennej **bmi**
- BMI obliczamy jako wynik dzielenia naszych argumentów, waga przez drugą potęgę wzrostu

Obliczanie BMI - objaśnienie

```
1 def calculate_bmi(weight, height):  
2     bmi = weight / height ** 2  
3     if bmi < 18.5:  
4         result = 'underweight'  
5     elif bmi > 25:  
6         result = 'overweight'  
7     else:  
8         result = 'normal'  
9     return result  
  
10 if __name__ == '__main__':  
11     user_weight = float(input('Weight in [kg]: '))  
12     user_height = float(input('Height in [m]: '))  
13     user_result = calculate_bmi(user_weight, user_height)  
14     print(f'Your BMI is {user_result}')
```

- Blok instrukcji warunkowej rozpoczętym znacznikiem **if**
- Po nim następuje wyrażenie logiczne z operatorem porównania, którego sprawdzenie nas interesuje
- Warunek kończymy dwukropkiem.
Wcięcie zawiera kod wykonywany w przypadku gdy sprawdzany warunek jest prawdziwy
- Pozostałe znaczniki **elif** i **else** są sprawdzane w przypadku gdy poprzedni warunek nie był spełniony

Obliczanie BMI - objaśnienie

```
1 def calculate_bmi(weight, height):
2     bmi = weight / height ** 2
3     if bmi < 18.5:
4         result = 'underweight'
5     elif bmi > 25:
6         result = 'overweight'
7     else:
8         result = 'normal'
9     return result
10
11 if __name__ == '__main__':
12     user_weight = float(input('Weight in [kg]: '))
13     user_height = float(input('Height in [m]: '))
14     user_result = calculate_bmi(user_weight, user_height)
15     print(f'Your BMI says {user_result}' )
```

- Słowo kluczowe **return** zwraca wartość wyznaczoną w trakcie wykonywania funkcji i kończy jej działanie
- Nie wszystkie funkcje muszą zwracać wartość

Obliczanie BMI - objaśnienie



```
1 def calculate_bmi(weight, height):
2     bmi = weight / height ** 2
3     if bmi < 18.5:
4         result = 'underweight'
5     elif bmi > 25:
6         result = 'overweight'
7     else:
8         result = 'normal'
9     return result
10
11 if __name__ == '__main__':
12     user_weight = float(input('Weight in [kg]: '))
13     user_height = float(input('Height in [m]: '))
14     user_result = calculate_bmi(user_weight, user_height)
15     print(f'Your BMI is {user_result}')
```

- Punkt wejściowy skryptu znajduje się w instrukcji warunkowej
- Kod znajdujący się za wcięciem będzie wykonany wyłącznie gdy pliku go zawierający zostanie bezpośrednio wykonany
- Jest to zabezpieczenie przed wykonaniem kodu podczas importowania pliku w innym module/pliku
- Do sprawdzenia warunku używamy operatora porównania ==

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Obliczanie BMI - objaśnienie



```
1 def calculate_bmi(weight, height):
2     bmi = weight / height ** 2
3     if bmi < 18.5:
4         result = 'underweight'
5     elif bmi > 25:
6         result = 'overweight'
7     else:
8         result = 'normal'
9     return result
10
11 if __name__ == '__main__':
12     user_weight = float(input('Weight in [kg]: '))
13     user_height = float(input('Height in [m]: '))
14     user_result = calculate_bmi(user_weight, user_height)
15     print(f'Your BMI says {user_result}')
```

- Przypisujemy do zmiennych wartości wprowadzane przez użytkownika
- Na ich podstawie obliczamy żądaną wartość BMI

- Wypisujemy wynik dodając wynik do ciągu znaków przy pomocy *f-string*

Obliczanie BMI – składnia języka

```
1 def calculate_bmi(weight, height):
2     bmi = weight / height ** 2
3     if bmi < 18.5:
4         result = 'underweight'
5     elif bmi > 25:
6         result = 'overweight'
7     else:
8         result = 'normal'
9     return result
10
11
12 if __name__ == '__main__':
13     user_weight = float(input('Weight in [kg]: '))
14     user_height = float(input('Height in [m]: '))
15     user_result = calculate_bmi(user_weight, user_height)
16     print(f'Your BMI says {user_result}')
```

- Wielkość liter nazw ma znaczenie w pythonie, zmienią *Result* i *result* to dwie różne zmienne
- Należy unikać krótkich, małoznaczących nazw zmiennych i funkcji
- Podwójny podkreśnik to *dunder* (**double underscore**)
- Funkcje, które zawierają dunder nazywamy specjalnymi (*__name__*)

Obliczanie BMI - shebang

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def calculate_bmi(weight, height):
    bmi = weight / height ** 2
    if bmi < 18.5:
        result = 'underweight'
    elif bmi > 25:
        result = 'overweight'
    else:
        result = 'normal'
    return result, bmi

if __name__ == '__main__':
    user_weight = float(input('Weight in [kg]: '))
    user_height = float(input('Height in [m]: '))
    user_result, user_bmi = calculate_bmi(user_weight, user_height)
    print(f'Your BMI: {user_bmi:.2f} says {user_result}.')
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Formatowanie tekstu



- Wyróżniamy kilka sposobów formatowania tekstu:
 - % - niezalecany
 - .format
 - f-string

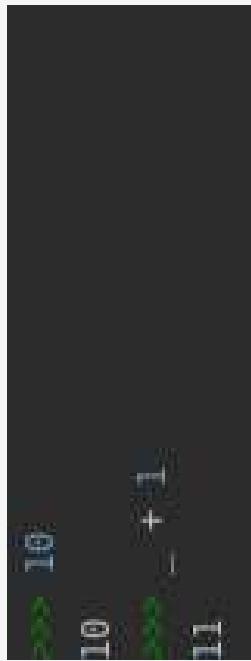
```
>>> '%s %% (%s)' % ('one', 'two')
'one two'
>>> 'Value: %03.2f', rest: {1:.2f}.format(1, 2)
'Value: 1, rest: 2.00'
>>> a = 1
>>> f'Value: {a}'
'Value: 1'
>>> b=2
>>> f'Letter: {b.capitalize()}' 
'Letter: A'
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Ciekawostka

- Jak to możliwe?



Cykl życia zmiennej

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  number = 2
4  power = 2
5
6
7  def calculate_power(value, exponent):
8      result = value ** exponent
9      return result
10
11
12 if __name__ == "__main__":
13     print(calculate_power(number, power))
14
```

- Zmienne w zależności od miejsca deklaracji posiadają odpowiedni zakres występowania
- Zmienna **number** i **power** to zmienne globalne
- Zmienna **result** to zmienna lokalna
- PyCharm posiada rozbudowany tryb debugera do pracy krokowej

Podstawowe typy zmiennych

- **Integer** (liczby całkowite)
- **Float** (liczby zmiennoprzecinkowe)
- **String** (teksty –łańcuchy znaków)
- **Boolean** (wartości logiczne -True/False, prawda/fałsz)
- **None** (specjalny typ oznaczający brak wartości)
- **List** (uporządkowane listy)
- **Tuple** (krotki)
- **Set** (zbiory)
- **Dict** (słowniki)

```
1 foo = 1
2 foo = 1.2
3 foo = "bar"
4 foo = False
5 foo = None
6 foo = [1, 2, 3]
7 foo = {1, 2, 3}
8 foo = {"spam": 1, "eggs": 2}
```

Foo bar



- W programowaniu zmiennym trzeba nadać jakieś nazwy i często zastanawiamy się jaka nazwa jest sensowna
- Czasami zdarza się, że chcemy wytlumaczyć jakąś koncepcję, podać jakiś przykład w którym nazwa zmiennej jest nieistotna, ważna jest idea, której chcemy wytlumaczyć
- Najbardziej popularnymi zwyczajowymi nazwami są **foo**, **bar** oraz **baz**
- **spam** i **eggs** są to nazwy bezpośrednio zaczerpnięte z programu Latający Cyrk Monty Pythona

Funkcje wbudowane

- Funkcje które dostępne są domyślnie, nie ma potrzeby ich importowania
- Matematyczne:
 - abs() – zwraca wartość absolutną liczby
 - max() – zwraca wartość maksymalną zbioru
- Konwersji typów:
 - int() – zwraca wartość całkowitą liczby
 - hex() – zwraca wartość heksadecymalną liczby
- Wejścia / wyjścia:
 - input() – odczytuje wprowadzane dane z konsoli
 - print() – wypisuje łańcuch znaków
- Referencji:
 - dir() – listuje właściwości obiektu
 - type() – zwraca typ obiektu

```
abs(-4)
max([1, 2, 3])
int(1.2)
hex(10)
'0xa'
```

Obiekty

- Wszystko w Pythonie jest obiektem
- Możemy korzystać z wbudowanych w obiekty metody

```
>>> isinstance('Text', object)
```

```
True
```

```
>>> isinstance(1, object)
```

```
True
```

```
>>> isinstance(None, object)
```

```
True
```

```
>>> isinstance(isinstance, object)
```

```
True
```

```
>>> help(object)
```

```
Help on class object in module builtins:
```

```
class object
|   The most base type
```

```
1  foo = 1
2  foo = 1.2
3  foo = 'bar'
4  foo = Fa[m capitalize(self)
5  foo = No[m casefold(self)
6  foo = [l[m center(self, width, fillchar)
7  foo = (l[m count(self, x, start, end)
8  foo = {l[m encode(self, encoding, errors)
9  foo = {'[m endswith(self, suffix, start, end)
10 foo = {'[m expandtabs(self, tabsize)
11 foo = {'[m find(self, sub, start, end)
12 foo = {'[m format(self, args, kwargs)
13 foo = {'[m format_map(self, map)
14 foo = {'[m index(self, sub, start, end)
15 foo = {'[m join(self, sep)
16 foo = {'[m ljust(self, width)
17 foo = {'[m replace(self, to_replace, replacement)
18 foo = {'[m rjust(self, width)
19 foo = {'[m rindex(self, sub, start, end)
20 foo = {'[m rpartition(self, sep)
21 foo = {'[m rsplit(self, sep, maxsplit)
22 foo = {'[m split(self, sep, maxsplit)
23 foo = {'[m strip(self, chars)
24 foo = {'[m swapcase(self)
25 foo = {'[m title(self)
26 foo = {'[m upper(self)
27 foo = {'[m zfill(self, width)

Press Enter to insert, Tab to replace
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Operatory



- Pozwalają manipulować wartościami zmiennych
- Python wspiera następujące typy operatorów:
 - Arytmetyczne
 - Porównania
 - Przypisania
 - Logiczne
 - Bitowe
 - Przynależności do zbioru
 - Stanu obiektu
- Kolejność wykonywania operatorów

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Operatory arytmetyczne

Operator	Opis
+	Dodawanie
-	Odejmowanie
*	Mnożenie
/	Dzielenie
%	Reszta z dzielenia
**	Potęgowanie
//	Dzielenie całkowite



Operatory porównania

Operator	Opis
<code>==</code>	Równość
<code>!=</code>	Różne
<code>></code>	Większy niż
<code><</code>	Mniejszy niż
<code>>=</code>	Większy lub równy
<code><=</code>	Mniejszy lub równy



Operatory przypisania

Operator	Opis
=	Przypisanie
+=	Dodanie wartości i przypisanie
-=	Odejście wartości i przypisanie
*=	Pomnożenie wartości i przypisanie
/=	Podzielenie wartości i przypisanie
%=	Operacja modulo i przypisanie
**=	Potęgowanie wartości i przypisanie
//=	Dzielenie całkowite i przypisanie

```
2 += 3  
File "<input>", Line 1  
SyntaxError: can't assign to literal  
    a = 2  
    a += 2  
    a  
    a  
    a  
4    a *= 2  
    a  
    a  
16
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Operatory bitowe



Operator	Opis
&	Koniuunkcja bitowa
	Alternatywa bitowa
^	Alternatywa bitowa rozłączna
~	Negacja bitowa
<<	Przesunięcie bitowe w lewo
>>	Przesunięcie bitowe w prawo

```
a = 0b1010
>>> b = 0b0110
>>> bin(a & b)
'0b10',
>>> a & b
2
>>> bin(a | b)
'0b1110'
>>> bin(a ^ b)
'0b1100'
>>> bin(~b)
'-0b111',
>>> 0b0001 << 1
2
>>> bin(0b0001 << 1)
'0b10',
```

Operatory logiczne

Operator	Opis
and	Koniunkcja
or	Alternatywa
not	Negacja



Operatory przynależności

Operator	Opis
in	Element zawiera się w sekwencji
not in	Element nie zawiera się w sekwencji

```
>>> 1 in [1, 2, 3]
True
>>> 4 in [1, 2, 3]
False
>>> 4 not in [1, 2, 3]
True
```

Operatory stanu obiektu

Operator	Opis
is	Zmienne wskazują na ten sam obiekt (id)
is not	Zmienne nie wskazują na ten sam obiekt

```
b = 1  
b = 2  
c = 15  
a is b  
True  
a is c  
False  
b = 4  
a is b  
False  
id(a)  
10068800
```

Operatory



- Możliwe jest także składanie operatorów w celu sprawdzenia bardziej zaawansowanych warunków

```
>>> (1.22 > 3) or (5 % 2 > 0) and (e in 'Text')
True
```

Obliczanie silni



- Iloczyn wszystkich liczb naturalnych dodatnich nie większych od podanej
- Oznaczane jako $n!$
- Oblicz silnie dowolnej liczby naturalnej

$n!$

Obliczanie silni

- Pojawiła się tutaj funkcja, która wywołuje samą siebie – poznaliśmy rekurencję

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4
5 def factorial(x):
6     if x == 0:
7         return 1
8     else:
9         return x * factorial(x-1)
10
11
12 if __name__ == '__main__':
13     value = 6
14     result = factorial(value)
15     print(F'Factorial of {value} is {result}' )
16
```

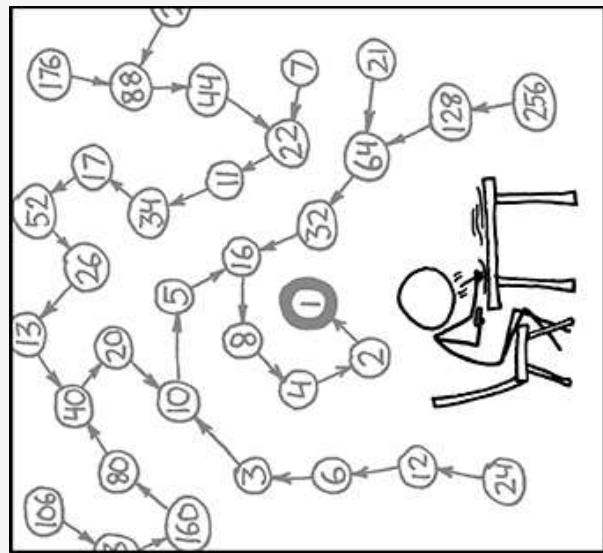
Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy



Rekurencja raz jeszcze – problem Collatza

- Weź dowolną liczbę naturalną.
- Jeśli jest parzysta, podziel ją przez 2.
- Jeśli jest nieparzysta pomnóż ją przez 3 i dodaj 1.
- Jeśli jest jedynką to skończ.
- Powyższy program zawsze w końcu dojdzie do jedynki dla dowolnej liczby całkowitej ale matematycy do dziś nie są w stanie udowodnić dlaczego.



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Rekurencja raz jeszcze – problem Collatza

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4
5  def collatz(number):
6      print(number)
7      if number == 1:
8          return
9      elif number % 2:
10         return collatz(number * 3 + 1)
11     else:
12         return collatz(number // 2)
13
14
15  if __name__ == '__main__':
16      user_number = int(input('Input a number: '))
17      print(f'Collatz sequence for number {user_number} is: ')
18      collatz(user_number)
19
```

Funkcje raz jeszcze

- Typowanie parametrów
- Typowanie wartości zwracanej
- Wywołanie z nazwą parametru
- Domyslna wartość parametru
- Możliwość podania parametru opcjonalnego przez podanie wartości domyslnej `None`

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  def factorial(x: int = 3) -> int:
5      if x == 0:
6          return 1
7      else:
8          return x * factorial(x-1)
9
10
11 if __name__ == '__main__':
12     value = 6
13     result = factorial(x=value)
14     print(f'Factorial of {value} is {result}')
15
16 result = factorial()
17 print(f'Factorial of default is {result}')
18
```

```
Factorial of 6 is 720
Factorial of default is 6
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Funkcje raz jeszcze

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  a_list = [1, 2, 3, 4]
5  b_list = [1, 2, 3, 4, 5, 6]
6
7  def sum_list(*args) -> float:
8      return sum(*args)
9
10
11  if __name__ == '__main__':
12      result = sum_list(a_list)
13      print(f'Sum of {a_list} is: {result}')
14      result = sum_list(b_list)
15      print(f'Sum of {b_list} is: {result}')
16
17
```

- Zmienna liczba argumentów
- Założeniem jest otrzymanie na wejściu listy (a właściwie elementu iterowalnego)

```
Sum of [1, 2, 3, 4] is: 10
Sum of [1, 2, 3, 4, 5, 6] is: 21
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Struktury danych - listy

- Najbardziej podstawową strukturą danych jest lista
- Każdy element listy posiada przypisany adres / indeks
- Indeksy rozpoczynamy od 0
- Indeksy mogą być ujemne, wybieramy wtedy elementy od końca listy
- Elementem listy może być obiekt
- Elementy listy mogą być różnego typu

```
a_list = [1, 2, 3, 4, 5, 6]
          a_list[-1]
6          a_list[-2]
5
```

```
a
[1, 2]
type(a)
<class 'List'>
```

Listy



- Tworzenie listy następuje poprzez podanie elementów rozdzielonych przecinkami w nawiasach kwadratowych
- Aktualizacja dowolnego pola realizowana jest przez przypisanie wartości pod dany indeks
- Usunięcie elementu wykonujemy za pomocą znacznika **del**

```
>>> a_list = [1, 'abc', True, 2]
>>> a_list[0]
1
>>> a_list[1:3]
['abc', True]
>>> a_list[2] = 0.5
>>> a_list
[1, 'abc', 0.5, 2]
>>> del a_list[0]
>>> a_list
['abc', 0.5, 2]
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Listy – podstawowe operacje

Wyrażenie	Opis
<code>len([1, 2, 3])</code>	Długość
<code>[1, 2, 3] + [4, 5, 6]</code>	Łączenie list
<code>["A"] * 4</code>	Powtarzanie list
<code>3 in [1, 2, 3]</code>	Przynależność do listy

```
len([1, 2, 3])
[1, 2, 3] + [4, 5, 6]
["A"] * 4
3 in [1, 2, 3]
```

Listy – szatkowanie

- Listy umożliwiają wydzielenie ich fragmentów
- Odbiera się to przez podanie w nawiasach liczb oddzielonych dwukropkiem, czyli podanie zakresu
- Kolejne liczby oznaczają początek i koniec wycinka oraz krok postępu, domyślnie o wartości 1

```
a_list = [1, 2, 3, 4, 5, 6]
a_list[1:4]
[2, 3, 4]
a_list[1:5:2]
[2, 4]
a_list[::1]
[6, 5, 4, 3, 2, 1]
a_list[::-1]
[1, 2, 3, 4, 5, 6]
a_list[1:]
[2, 3, 4, 5, 6]
a_list[:-1]
[1, 2, 3, 4, 5]
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Listy – metody

Metoda	Opis
.append(obj)	Dopisanie elementu obj na końcu listy
.count(obj)	Zliczenie wystąpień elementu obj
.extend(seq)	Dopisanie sekwencji do listy
.index(obj)	Najniższy indeks elementu obj
.pop()	Usuwa i zwraca ostatni element z listy
.remove(obj)	Usuwa element obj z listy
.reverse()	Zwraca listę w odwrotnej kolejności
.sort([func])	Sortuje listę z możliwością podania metody
sorted(seq)	Zwraca posortowaną listę

```
>>> a_list = [1, 2, 3, 4, 5, 6]
>>> a_list
[1, 2, 3, 4, 5, 6]
>>> a_list.append(object)
>>> a_list
[1, 2, 3, 4, 5, 6, object]
>>> a_list.clear()
>>> a_list
[]
>>> a_list.copy()
>>> a_list
[1, 2, 3, 4, 5, 6]
>>> a_list.count(value)
>>> a_list.extend(iterable)
>>> a_list
[1, 2, 3, 4, 5, 6, value]
>>> a_list.index(value)
>>> a_list
[1, 2, 3, 4, 5, 6]
>>> a_list.insert(index, object)
>>> a_list
[1, 2, 3, 4, 5, 6, object]
>>> a_list.pop()
>>> a_list
[1, 2, 3, 4, 5]
>>> a_list.remove(value)
>>> a_list
[1, 2, 3, 4]
>>> a_list.reverse()
>>> a_list
[6, 5, 4, 3, 2, 1]
>>> a_list.sort(key=None, reverse=False)
>>> a_list
[1, 2, 3, 4, 5, 6]
Press ctrl+ to choose the selected (or fini
>>> a_list.
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Listy – rozpakowanie

- Elementy listy można przypisać do zmiennych nie tylko odnosząc się do indeksów elementów
- Python posiada ciekawy mechanizm rozpakowywania listy na zmienne

```
a_list = [1, 2, 3]
a, b, c = a_list
x, y = a_list
x
1 2 3
x
1 [2, 3]
```



Funkcja wbudowana `zip`

- Jako parametry funkcja przyjmuje dwie lub więcej list
- Wynikiem działania jest lista kolejnych elementów, spарowanych według indeksu z każdej z list na wejściu funkcji
- Jeśli listy mają różne długości, to wynik będzie miał długość najkrótszej z nich

```
>>> a_list = [1, 2, 3, 4]
>>> b_list = ['a', 'b', 'c', 'd']
>>> zip(a_list, b_list)
<zip object at 0x7f8a312a9e08>
>>> list(zip(a_list, b_list))
[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd')]
```

Funkcja wbudowana range

- Jedną z ostatnich ważnych dla nas funkcji jest **range** (ściśle biorąc nie jest to funkcja ale my możemy ją tak traktować).
- **range** zwraca sekwencję liczb całkowitych.
- **range(n)** zwraca sekwencję liczb od 0 do n-1.
- **range(a, b)** zwraca sekwencję liczb od a do b-1.
- **range(a, b, c)** zwraca sekwencję liczb [a, a+c, a+2c, ...] aż do b-1.
- Używając funkcji range można szybko tworzyć listy.

```
list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Pętla for

- Pozwala wykonać zaplanowaną operację określonaą liczbę razy

```
def for_hello(name):  
    for i in range(3):  
        print(f'Hello {name}!')  
  
for_hello('Python')  
Hello Python!  
Hello Python!  
Hello Python!
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Pętla for

- W Pythonie często wykorzystujemy pętlę **for** do iterowania po elementach list

```
for x in 'Python':  
    print(f'Current x value: {x}')  
  
Current x value: P  
Current x value: y  
Current x value: t  
Current x value: h  
Current x value: o  
Current x value: n
```

Pętla for z indeksem

- Oprócz wartości elementu możemy pobrać także jego indeks w danej liście

```
>>> for index, value in enumerate('Python'):
...     print(f'Current value: {value} on index: {index}')
...
Current value: P on index: 0
Current value: y on index: 1
Current value: t on index: 2
Current value: h on index: 3
Current value: o on index: 4
Current value: n on index: 5
```

Pętla `for` z `else`?

- Po wyczerpaniu elementów lub braku elementów sekwencji, po której iterujemy możemy wykonać osobny blok kodu

```
>>> data = range(3)
>>> for x in data:
...     print(f'Current x value: {x}')
... else:
...     print('End of string')
...
Current x value: 0
Current x value: 1
Current x value: 2
End of string
>>> data = []
>>> for x in data:
...     print(f'Current x value: {x}')
... else:
...     print('End of string')
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Pętla while

- Pętla **while** działa tak długo jak długo spełniony jest logiczny warunek podany w jej wywołaniu

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4
5 def count_down(number):
6     while number:
7         print(number)
8         number -= 1
9         print('Now we know while loop!')
10
11 if __name__ == '__main__':
12     count_down(10)
13
14
```

```
10
9
8
7
6
5
4
3
2
1
Now we know while loop!
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy

Pętle – słowa kluczowe **break** i **continue**

- Słowo kluczowe **break** przerwuje wykonanie pętli, nawet jeśli warunek pętli jest spełniony.
- Słowo kluczowe **continue** przerwuje iterację i przechodzi do następnego obiegu pętli.

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4
5  def count_down(number):
6      while number:
7          number -= 1
8          if number % 2:
9              print(number)
10             continue
11         print('Now we know while loop!')
12
13
14  if __name__ == '__main__':
15      count_down(10)
16
17
```

```
1  Now we know while loop!
2
3
4
5  Now we know while loop!
```

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy



Kolejny program – testy DNA

- Ojcostwo ustala się przy pomocy badań genetycznych.
- Materiał genetyczny potencjalnych ojców jest porównywany z materiałem dziecka.
- Za ojca uznaje się osobę, której materiał genetyczny jest najbardziej podobny do dziecka.
- Materiał genetyczny to nić DNA - ciąg nukleotydów (znaków składających się z czterech liter: A, C, T oraz G).
- Aby obliczyć podobieństwo dwóch nici DNA używa się tzw. odległości Hamminga.
- Polega to na tym, że patrzy się na kolejne litery dwóch nici - za każdą niezgodność liter przyznaje się jeden punkt. Odległość jest sumą punktów po całej nici.
- DNA ojca ma najmniejszą odległość Hamminga spośród wszystkich kandydatów.
- Oczywiście to bardzo uproszczony model i tak się tego nie robi!

Autor: Łukasz Kordowski

Prawa do korzystania z materiałów posiada Software Development Academy



Kolejny program – testy DNA

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 suspect_1 = 'GAGCCCTACTAAACGGGGAT'
5 suspect_2 = 'GAGCCCTACTAAACAAAT'
6 child = 'CATCGTAATTGACCGCCCT'
7
8 def hamming(strand_a, strand_b):
9     result = 0
10    zipped_strands = zip(strand_a, strand_b)
11    for item_a, item_b in zipped_strands:
12        if item_a != item_b:
13            result += 1
14    return result
15
16
17 def __name__ == '__main__':
18     print(f'Suspect #1: {suspect_1}')
19     print(f'Suspect #2: {suspect_2}')
20     print(f'Child: {child}')
21
22     distance_1 = hamming(suspect_1, child)
23     distance_2 = hamming(suspect_2, child)
24     if distance_1 < distance_2:
25         print('Suspect #1 is a father')
26     else:
27         print('Suspect #2 is a father')
28
29
30 Suspect #1: GAGCCCTACTAAACGGGGAT
31 Suspect #2: GAGCCCTACTAAACAAAT
32 Child: CATCGTAATTGACCGCCCT
33 Suspect #1 is a father
```

Autor: Łukasz Kordowski
Prawa do korzystania z materiałów posiada Software Development Academy

- Na początku odległość wynosi 0.
- Łączymy w pary litery na obu niciach.
- Dla każdej pary sprawdzamy czy jej elementy są różne.
- Jeśli tak to zwiększamy dystans o jeden.
- W ten sposób liczymy odległość pomiędzy DNA dziecka i obu potencjalnych ojców.
- Za ojca uznajemy tego, którego odległość od DNA dziecka jest mniejsza.

```
Suspect #1: GAGCCCTACTAAACGGGGAT
Suspect #2: GAGCCCTACTAAACAAAT
Child: CATCGTAATTGACCGCCCT
Suspect #1 is a father
```