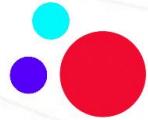




# Edytory kodu, IDE

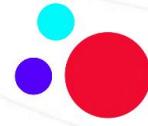
infoShare Academy

[infoShareAcademy.com](http://infoShareAcademy.com)



# Tworzenie kodu

- w interpreterze
- w plikach tekstowych (.py, .pyc)
  - edytor tekstowy
  - IDE – środowisko zawierające dodatkowe funkcje
    - Jupyter Notebook
    - PyCharm
    - Spyder

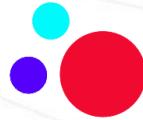


# **PyCharm – wywołanie tylko zaznaczonych linii**

Alt + Shift + E

# Struktura kodu Pythona

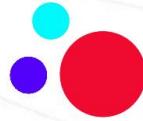
infoShare Academy



# Struktura kodu Pythona

Wygląd kodu ma znaczenie:

- znak nowej linii jest końcem instrukcji
- wcięcia wydzielają bloki kodu (mają znaczenie, nie możemy ich stosować dowolnie)
  - 4 spacje albo tab (nie można stosować zamiennie – trzeba zdecydować się na jeden sposób [4 spacje są ogólnoprzyjęte])



# Struktura kodu Pythona – wcięcia

Instrukcja 1

Instrukcja 2

Instrukcja/wyrażenie 3:

Instrukcja 3.1

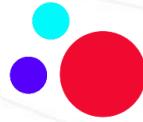
Instrukcja 3.2:

Instrukcja3.2.1

Instrukcja 4

Instrukcja 5

Instrukcja 6



# Struktura kodu Pythona – komentarze

```
Instrukcja 1  
# komentarz  
Instrukcja 2  
Instrukcja 3  
Instrukcja 4 # komentarz "in-line"
```



# Zmienne

infoShare Academy

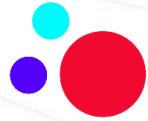
[infoShareAcademy.com](http://infoShareAcademy.com)



## Zmienne - podstawy

Zmienna to zarezerwowany obszar pamięci, który pozwala przechowywać jakąś wartość.

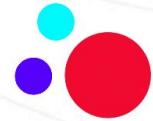
- używając nazwy zmiennej, możemy wykorzystywać ją wielokrotnie w kodzie
- zmienne mogą być poddawane operacjom i być argumentami funkcji



# Zmienne w Pythonie – wskaźniki

Zmienna to zarezerwowany obszar pamięci, który pozwala przechowywać jakąś wartość.

- zmienne w Pythonie to wskaźniki do miejsc w pamięci
- zmienne w Pythonie są dynamicznie typowane
  - nie musimy deklarować typu zmiennej przed jej przypisaniem
  - zmienna może zmienić swój typ

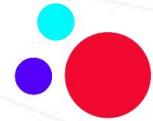


# Zmienne w Pythonie – wskaźniki

- Dwie zmienne odwołujące się do tego samego miejsca w pamięci

```
foo = 1  
bar = foo
```



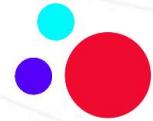


# Zmienne w Pythonie – wskaźniki

- Dwie zmienne odwołujące się do tego samego miejsca w pamięci

```
foo = 1  
bar = foo  
id(foo)  
id(bar)
```





# Zmienne w Pythonie – wskaźniki

- Dwie zmienne odwołujące się do tego samego miejsca w pamięci

```
foo = 1  
bar = 2  
id(foo)  
id(bar)
```

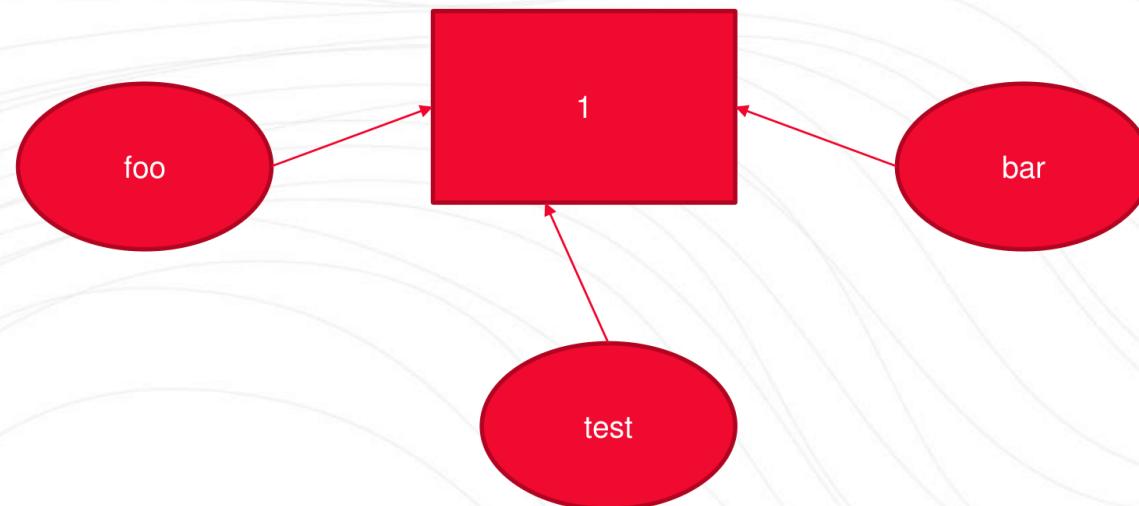


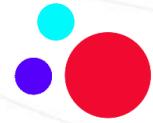


# Zmienne w Pythonie – wskaźniki

- Trzy zmienne odwołujące się do tego samego miejsca w pamięci – ten przypadek omówimy za tydzień :)

```
foo = 1
bar = foo
test = 1
id(foo)
id(bar)
id(test)
```





# Zmienne – podstawy

Przypisanie wartości do zmiennej:

```
foo = 123
```

- `foo` – nazwa zmiennej
- `123` – wartość zmiennej (int 123)
- po prawej stronie `=` możemy mieć całe wyrażenie np. `2 + 2`

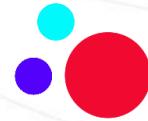


# Zmienne - podstawy

Przypisanie tej samej wartości do kilku zmiennych:

```
foo = bar = a = b = 23
```

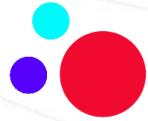
```
print('Zmienna ', foo)
print('Zmienna ', bar)
print('Zmienna ', a)
print('Zmienna ', b)
```



## Zmienne – odwołanie się do wartości

Odwołanie do zmiennej polega na użyciu jej nazwy:

```
print(zmienna)
```



## Zmienne – odwołanie się do siebie samej

Zmienna może odwołać się sama do siebie np. po to, żeby zmienić swoją wartość:

```
foo = 1
```

```
print(foo)
```

```
foo = foo + 2
```

```
print(foo)
```



# Zmienne – zasady nazewnictwa

Nazwy zmiennych:

- powinny mieć sens – niech nazwa zmiennej coś mówi osobie czytającej nasz kod
- nie mogą zawierać znaków specjalnych
- nie mogą zaczynać się od cyfr
  - ale mogą zawierać cyfry
- powinny trzymać się jednej konwencji



# Zmienne – naming conventions

Naming conventions:

- snake\_case (poleczany sposób nazewnictwa zmiennych)

```
monty_python = 1
```

- MACRO\_CASE (lepiej nie)

```
MONTY_PYTHON = 1 # wielkimi literami w Pythonie oznacza się stałe
```

- camelCase

```
montyPython = 1
```

- CapWord

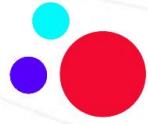
```
MontyPython = 1
```



# Typy danych

infoShare Academy

[infoShareAcademy.com](http://infoShareAcademy.com)



# Typy danych vs struktury danych

Typy danych (**data types**):

- klasa konkretnych obiektów, których pewne właściwości są te same (np. liczby całkowite)

Struktury danych (**data structures**):

- opis sposobu organizacji danych (np. lista)

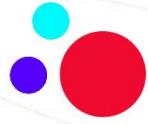
Ten podział jest jednak raczej teoretyczny, w praktyce <https://docs.python.org/3/library/datatypes.html>



# Funkcje – podstawy

Funkcja to nazwany fragment kodu, który możemy wykorzystywać wielokrotnie w różnych miejscach w programie np. print()

- przyjmuje argumenty, które decydują o jej wyniku np. print("argument")
  - idealnie funkcja dla tych samych argumentów powinna zawsze zwrócić ten sam wynik
- w Pythonie istnieją wbudowane funkcje (np. print() )
  - <https://docs.python.org/3/library/functions.html>

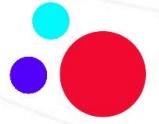


# Funkcja type()

Wbudowana funkcja Pythona

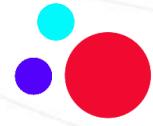
- przy podaniu jednego argumentu zwraca jego typ

```
>>type(24)  
int
```



# Typy danych w Pythonie

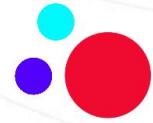
- **int** – liczby całkowite (integers) np. 2435
- **float** – liczby zmiennoprzecinkowe (floating-point numbers) np. 1.25
- **complex** – liczby zespolone (complex numbers) np. 4+1j
- **str** – łańcuchy znaków (string) np. "Monty Python"
- **bool** – prawda / fałsz (True / False)
- **None** – pusty obiekt; nic



# Typy danych – liczby

Chociaż matematycznie każda z tych liczb to liczba całkowita 1, to sposób zapisu determinuje jak Python ją interpretuje:

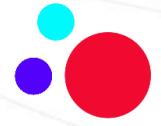
- `1` – int
- `1.0` – float
- `1+0j` – complex



# Typy danych – liczby

Również tutaj dla Pythona to nie jest 1, ale ciąg znaków:

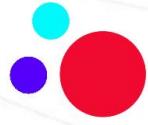
- "1" – string
- '1' – str
- """1""" – str



# Typy danych – łańcuchy znaków (strings)

Łańcuchy znaków możemy zapisać na 3 sposoby:

- "Monty Python"
- 'Monty Python'
- """Monty Python"""



# Typy danych – łańcuchy znaków (strings)

Łańcuchy znaków możemy zapisać na 3 sposoby:

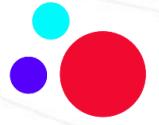
```
"""Monty  
Python"""
```



# Typy danych – łańcuchy znaków (strings)

Niektóre znaki mogą mieć specjalne znaczenie przy wypisywaniu stringów:

```
>>print("a\nb")  
a  
b
```

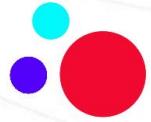


# Typy danych – boolean (bool)

Prawda i fałsz:

- True
- False

Dane typu bool podlegają rachunkowi logicznemu

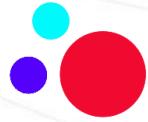


# Typy danych – None

Nic:

- None

(trochę jak NULL w SQL)



# Konwersja typów (rzutowanie)

Typy danych można konwertować w inne typy.

- nie zawsze jest to możliwe np. nie przekonwertujemy `1+3j` do integera
- czasem jest to konieczne, ponieważ nasz kod inaczej nie będzie chciał działać (np. nie pomnożymy dwóch liczb, które Python widzi jako stringi)



# Konwersja typów (rzutowanie)

Kilka konwersji

- `float()` – rzutuje do float
- `int()` – rzutuje do int
- `complex()` – rzutuje do complex
- `str()` – rzutuje do str
- `bool()` – rzutuje do bool



# Operatory

infoShare Academy

[infoShareAcademy.com](http://infoShareAcademy.com)



# Operatory matematyczne - podstawy

4 podstawowe operatory:

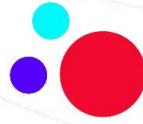
- $+$  - dodawanie
- $-$  - odejmowanie
- $*$  - mnożenie
- $/$  - dzielenie



# Operatory matematyczne - podstawy

Działania wykonywane są od lewej do prawej, z zachowaniem zasad matematyki

- można stosować ( ) w celu zmiany kolejności działań



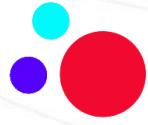
# Operatory matematyczne

- `%` - dzielenie modulo (reszta z dzielenia)
- `//` - dzielenie całkowite
- `**` - potęgowanie



# Operatory porównania

- `==` - czy równy?
- `!=` - czy różny?
- `>` i `<` - czy większy / mniejszy?
- `>=` i `<=` - czy większy-równy / mniejszy-równy



# Operatory logiczne – podstawowe

2 podstawowe operatory logiczne:

- **and** – koniunkcja (i)
- **or** – alternatywa (lub)



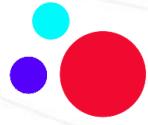
# Tabela wartości logicznych

A	B	A <b>and</b> B	A <b>or</b> B	A <b>xor</b> B
True	True	True	True	False
True	False	False	True	True
False	True	False	True	True
False	False	False	False	False



# Operatory bitowe

- `&` - binarne AND
- `|` - binarne OR
- `^` - binarne XOR
- `<<` - przesunięcie bitowe w lewo
- `>>` - przesunięcie bitowe w prawo



# Operatory przypisania

- **=** - przypisanie wartości
- **+=** - dodanie wartości do bieżącej
  - $a += 1$  #  $a = a + 1$
- **-=** - odejmowanie wartości od bieżącej
  - $a -= 1$  #  $a = a - 1$
- **\*=** - mnożenie wartości bieżącej
  - $a *= 1$  #  $a = a * 1$
- **/=** - dzielenie wartości bieżącej
  - $a /= 1$  #  $a = a / 1$



- **in / not in** – obecność w sekwencji / kolekcji (w stringu też)
- **is / is not** – porównanie lokalizacji w pamięci (lepszy **==**)

# Przyjmowanie danych od użytkownika

infoShare Academy



# Funkcja input()

- zatrzymuje działanie programu czekając na wprowadzenie inputu przez użytkownika i wciśnięcie klawisza ENTER
- przypisuje wprowadzony input do zmiennej

```
foo = input()
```



# Instrukcje warunkowe

infoShare Academy

[infoShareAcademy.com](http://infoShareAcademy.com)



```
if warunek:  
    # kod do wykonania gdy warunek prawdziwy
```



```
if warunek:  
    # kod do wykonania gdy warunek prawdziwy  
else:  
    # kod wykonany gdy if był fałszywy
```



```
if warunek:  
    # kod do wykonania gdy warunek prawdziwy  
elif inny warunek:  
    # kod wykonany gdy if był fałszywy, a ten elif jest prawdziwy  
# (...) więcej elif'ów  
else:  
    # kod wykonany gdy if i każdy elif był fałszywy
```



# Dzięki

<https://www.linkedin.com/in/piotr-klinke-ai/>

infoShareAcademy.com





