
infoShare
ACADEMY
warsztaty
Python
dla początkujących

Hello

Bartłomiej Biernacki

bartlomiej@biernacki.me

github.com/pax0r

Agenda

- 18:00-19:30 Podstawy Python
  - Dlaczego Python?
  - Podstawowe elementy języka
  - Podstawowe struktury danych
  - Obsługa plików
- ***19:30-19:45 Przerwa***
- 19:45-21:00 Warsztaty
  - Przykładowy projekt
  - Praca własna
  - Przykłady wykorzystania

Zasoby

- **Google**
- Dokumentacja Python: https://docs.python.org/3/
- StackOverflow: https://stackoverflow.com/
- GitHub: https://github.com/pax0r/python-warsztaty-2

1. Podstawy Python

Dlaczego Python?

- Prosta składnia (syntax)
- Kompaktowy kod
- Kod niezależny od systemu
- Wszechstronny (Big Data, AI, Web, devops, pentesting)
- Popularność (Facebook, Google, Instagram, Dropbox)

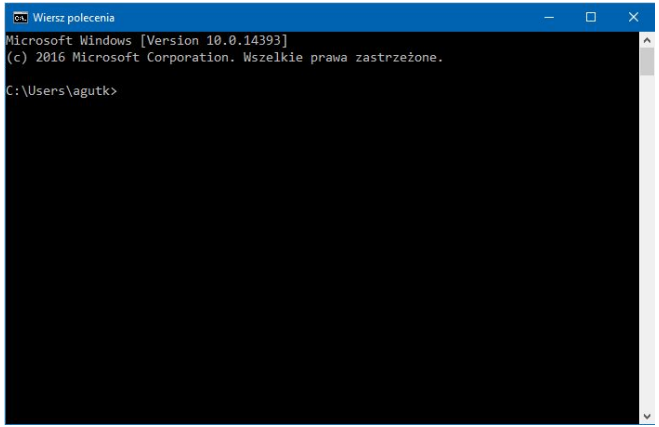
Tworzenie kodu

- interpreter
- zwykły notatnik (pliki tekstowe *.py)
- IDE - dodatkowa funkcjonalność takie jak:
  - podpowiedzi
  - kolorowanie składni,
  - debugger,
  - testy

Python IDLE, **PyCharm**, VS Code, Sublime, Atom

Uruchamianie kodu

- interpreter
- konsola/wiersz poleceń (*python.exe*)



- IDE umożliwiają uruchamianie bezpośrednio
  - nie zawsze program zadziała bez IDE!

Typy danych

- `123` - `int` – liczby całkowite
- `54.45` - `float` – liczby zmienno-przecinkowe
- `"Ala"` - `str` – łańcuchy znaków (string)
- `True/False` - `bool` – prawda fałsz
- `None` – nic, null, brak

listy, słowniki, tuple
pliki, własne typy

Zmienna

- nazwany obszar pamięci, w którym znajduje się jakaś wartość
- pozwala na ponowne użycie wartości w innym miejscu w kodzie

```
moja_liczba = 124
nazwisko = "Kowalski"
czy_obecny = True
```

= to jest znak przypisania

Operatory

Matematyczne:

`+, -, *, /, //, %, **`

Logiczne:

`==, !=, <, >, <=, >=, in, is, and, or, not`

Operator przypisania

=

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

```
wynik = 5 != 4 and 'a' not in 'Andrzej'  # wynik będzie True
```

Operator porównania

= VS ==

= przypisuje wartość do zmiennej

```
x = 1
```

== porównuje dwie wartości (zwraca True lub False)

```
1 == (2 - 1)  # zwróci True
```

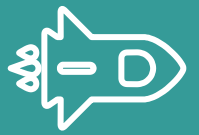
Komentarze

#

Wszystko po tym znaku jest ignorowane przez interpreter.

Może służyć do opisania fragmentu kodu.

Przykłady



Metody wbudowane typów

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

- `typ.funkcja()`
- `”ala ma kota”.capitalize()`

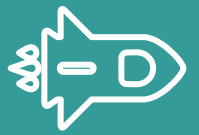
String

```
nazwisko = "Kowalski"
# długość
len(nazwisko) -> 8
# Indeksowanie
nazwisko[0] -> K
nazwisko[3] -> a
nazwisko[8] -> błąd, nie ma takiego indeksu!
```

int/float/string

`5`	- `int`	-	liczba całkowita
`65.987`	- `float`	-	liczba zmiennoprzecinkowa
`'45'`	- `str`	-	łańcuch znaków
`"3434.434"`	- `str`	-	łańcuch znaków

Przykłady



Funkcje input i print

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

`input()` przyjmuje od użytkownika dane i zapisuje do zmiennej. Wszystko jest stringiem.

```
print(nazwisko)
```

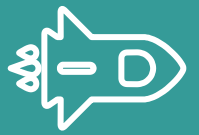
print() służy do wydrukowania tekstu na ekranie; automatycznie dodaje na końcu stringa znak specjalny nowej linii \n

Żółw

- Proste środowisko bazowane na popularnym “żółwiu” z Logo
- Wyświetla żółwia, któremu można wydawać komendy w stylu “idź do przodu”, “obróć się” itp.

```
import turtle          # uruchamia srodowisko zolwia
turtle.mainloop()      # czeka na interakcje
```

Przykłady



BLOK KODU

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

I tak dalej...
```

Dwukropek rozpoczynający blok

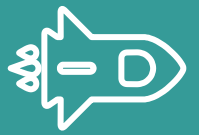
Indentacja 1 poziom (4 spacje)

Indentacja 2 poziom (8 spacji)

Instrukcja warunkowa

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

Przykłady



range

```
range(stop)
range(3)   -   <0, 1, 2>    //    len() == 3

range(start, stop)
range(4, 8)    -   <4, 5, 6, 7>

range(start, stop, krok)
range(0, 10, 3) – <0, 3, 6, 9>
```

Lista

list(), []

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

Możemy indeksować, slice'ować
Do elementu odwołujemy się przez indeks

Krotka

tuple()

Tuple jest typem niezmiennym – raz zdefiniowanego nie można zmienić

```
tuple1 = ("raz", "dwa", "trzy")

tuple1[0] = "jeden" - spowoduje błąd

x = "raz",
y = "raz", dwa"
```

Słownik

dict(), {}

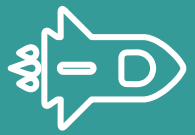
{klucz : wartość}

klucz – musi być typem niezmiennym (string, tuple, liczba), musi być unikalny (tylko jeden w słowniku)
wartość – mogą być powtórzone

Odwołujemy się poprzez klucz a nie indeks!!!

```
slownik = {“klucz”: “wartosc”}
print(slownik[“klucz”])
```

Przykłady



Pliki:

- `czesc_1/06_import.py`
- `czesc_1/07_range.py`
- `czesc_1/08_listy_tuple.py`
- `czesc_1/09_dict.py`

Pętla while

```
while (wartość logiczna True):
    kod
    ...
    update wartości logicznej na False
```

Kod wewnątrz pętli while, będzie powtarzany dopóki wartość logiczna (wyrażenia lub zmiennej) nie zmieni się na False*

* chyba, że pętla zostanie przerwana lub zmodyfikowana

Pętla for

```
for element in kolekcja:
    możemy użyć element
    ...
```

Pętla „for” wykona się tyle razy ile elementów jest w kolekcji*

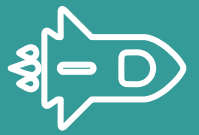
* chyba, że pętla zostanie przerwana lub zmodyfikowana

continue, break

`continue` – program pomija pozostałe instrukcje w bloku i wraca do sprawdzenia warunku (while) lub do kolejnego elementu (for)

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

Przykłady



Funkcje

definiowanie:

```
def do_nothing():
    pass
```

wywołanie:

```
do_nothing()
```

Argumenty funkcji

```
def do_nothing():                    # nie ma argumentów
    pass

def do_nothing(x):                   # jeden argument
    pass

def do_nothing(x, y, z):             # wiele argumentów
    pass
```

Argumenty domyślne

```
def do_nothing(x, y=10):
    pass

def do_nothing(x, y, z=12, w =„Ola”):
    pass

def do_nothing(y=10):
    pass
```

argumenty domyślne muszą być po argumentach wymaganych

argument domyślny jest sprawdzany tylko przy pierwszym wywołaniu funkcji – uwaga na typy referencyjne!

Argumenty domyślne

wywołanie

```
def do_something(x, y, z=12, w =„Ola”):
    pass

>>> do_something(1)        <- błąd – wszystkie arg. pozycyjne muszą być podane
>>> do_something(1, 23)
>>> do_something(1, 2, "trzy")
>>> do_something(1, 2, 34, "ola")
>>> do_something(1, 33, w="ola")
```

return

funkcja może robić coś wewnątrz siebie **(nawet nie trzeba print)**

```
def print_square(x)
    print(x**2)  # funkcja nic nie zwraca, wypisuje na ekran
```

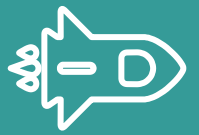
funkcja może oddać jakiś wynik/obiekt – używamy return

```
def give_square(x)
    return x**2  # funkcja nic nie wypisuje na ekran, zwraca wynik
```

aby użyć funkcję zwracającą obiekt należy ten obiekt zapisać w zmiennej

```
>>> wynik = give_square(3)
>>> print(wynik)
9
```

Przykłady



Obsługa plików

otwieramy plik

```
plik = open("sciezka_do_pliku", tryb)
```

tryby:
r – tylko do odczytu
w – zapisywanie pliku (stary plik o tej samej nazwie będzie usunięty)
r+ - do odczytu i zapisu
a – dopisywanie do pliku (dane są dopisane do końca istniejącego pliku)

Obsługa plików

pliki należy zamykac po użyciu:

```
plik = open(„plik.txt”)
    # kod
plik.close()
```

otwarcie pliku za pomocą `with` pozwala na automatyczne zamykanie pliku przez Pythona

```
with open(„plik.txt”) as  plik:
    print(plik.readline())
```

Obsługa plików

plik.read() – odczytanie całego pliku, zwracany jest string zawierający cały tekst pliku (włącznie ze znakami \n) – opc. argument – **int** określająca ilość bajtów do wczytania

plik.readline() – odczytanie jednej linii z pliku, zwracany jest string z linijką testu, włącznie ze znakiem \n

plik.readlines() – odczytuje cały tekst – zwraca listę stringów - linijek, włącznie ze znakiem \n

```
for line in plik:
    print(line, end='')
```

Obsługa plików

plik.read() – odczytanie całego pliku, zwracany jest string zawierający cały tekst pliku (włącznie ze znakami \n) – opc. argument – **int** określająca ilość bajtów do wczytania

plik.readline() – odczytanie jednej linii z pliku, zwracany jest string z linijką testu, włącznie ze znakiem \n

plik.readlines() – odczytuje cały tekst – zwraca listę stringów - linijek, włącznie ze znakiem \n

```
for line in plik:
    print(line, end='')
```

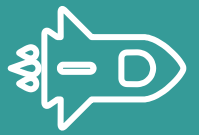
Obsługa plików

plik.write(string) – zapisuje string do pliku w obecnej pozycji kursora, zwraca liczbę zapisanych znaków – należy pamiętać o znaku \n

plik.writelines(iterable) – zapisuje elementy z kolekcji jako poszczególne linie w pliku - znak nowej lini powinien należeć do stringa!

Plik musi być otworzony w trybie do zapisu aby móc go zmieniać!

Przykłady



import

```
import moduł
from moduł import funkcja
from moduł import *
```

```
string, datetime, copy, math, decimal,
random, os, csv, antigravity
```

PRZERWA 15 MINUT

2. Warsztaty

Zadanie 1

- Wczytaj plik z repozytorium zawierający uczniów oraz oceny ./zadania/oceny.txt
- Znajdź osobę z najwyższą oraz najniższą oceną oraz średnią ocen całej klasy

Zadanie 2

- Wczytaj plik z repozytorium zawierający kroki żółwia
  ./zadania/zolw.txt
- Każda linijka zawiera jedną literę, spację oraz dwucyfrową liczbę
- Dla każdej linijki z tego pliku żółw powinien wykonać czynność odpowiadającą literze z podaną liczbą jako argumentem
- Komendy:
  - F - żółw naprzód (`turtle.forward(x)`)
  - B - żółw do tyłu (`turtle.backward(x)`)
  - L - obrót w lewo (`turtle.left(x)`)
  - R - obrót w prawo (`turtle.right(x)`)
- Przykładowo linijka zawierająca `F 40` - żółw idzie naprzód 40 jednostek (`turtle.forward(40)`)

Thanks!

bartlomiej@biernacki.me

github.com/pax0r