
infoShare
ACADEMY
warsztaty
Python
dla początkujących

Hello

Bartłomiej Biernacki

bartlomiej@biernacki.me

github.com/pax0r

Agenda

- `18:00-19:50` Podstawy Python
- *`19:50-20:00` Przerwa*
- `20:00-21:00` Aplikacja okienkowa

Zasoby

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

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)

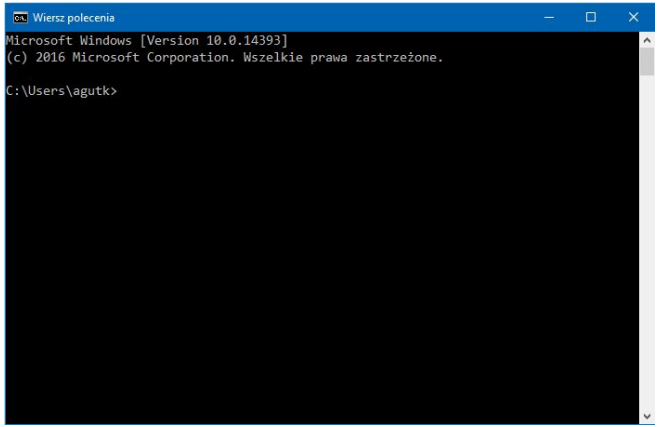
Tworzenie kodu

- 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
```

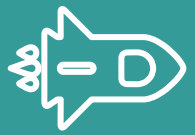
Komentarze

#

Wszystko po tym znaku jest ignorowane przez interpreter.

Może służyć do opisania fragmentu kodu.

Przykłady



Pliki:

- `czesc_1/00_typy.py`
- `czesc_1/01_zmienne.py`

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

Funkcje input i print

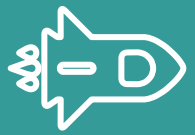
```
nazwisko = input(”Podaj nazwisko: ”)
```

`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

Przykłady



Pliki:

- `czesc_1/02_int_string.py`
- `czesc_1/03_input.py`

BLOK KODU

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

I tak dalej...
```

Dwukropek rozpoczynający blok

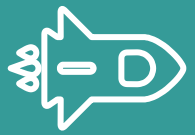
Indentacja 1 poziom (4 spacje)

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



Pliki:

- `czesc_1/04_if.py`
- `czesc_1/05_if.py`

import

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

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

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(), {}

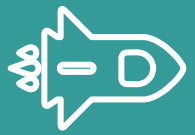
{klucz : wartość}

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*

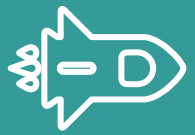
* chyba, że pętla zostanie przerwana lub zmodyfikowana

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



Pliki:

- `czesc_1/10_while.py`
- `czesc_1/11_for.py`

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)
```

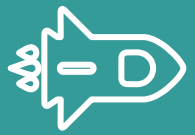
funkcja może oddać jakiś wynik/obiekt – używamy return

```
def give_square(x)
    return x**2
```

aby użyć funkcję zwracającą obiekt należy ten obiekt zapisać w zmiennej

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

Przykłady



Pliki:

- `czesc_1/11_funkcje.py`
- `czesc_1/12_funkcje.py`
- `czesc_1/13_funkcje.py`

Obiekty

```
1234    2.343534    'Magdalena'     [1, 3, 5, 7, 9]
{'imie': Andrzej', 'nazwisko': 'kowalski'}
```

dane ww. są instancjami obiektu, każdy obiekt ma:

- typ
- wewnętrzną reprezentację danych (prosta, złożona)
- zestaw procedur do interakcji z obiektem (in. interfejs)

Każda instancja jest konkretnym typem obiektu:

- **1234** jest instancją **int**
- `x = 'Natalia'` – **x** jest **instancją** instancją **string**
- **Azor** jest **instancją** klasy **Pies**

definiowanie klas

słowo kluczowe — nazwa — klasa nadrzędna / rodzic

```
class Samochod(object):
    # definicje danych
    # definicje metod
```

- **class** – podobnie jak **def**
- słowo **object** oznacza, że Samochód jest obiektem w Python (object) i **dziedziczy** z niego wszystkie właściwości
  - Samochod jest podklasą object
  - object jest klasą nadrzędną dla Samochod

Konstruktor

parametr – referencja instancji

dane inicjalizujące

```
class Samochod(object):
    def __init__(self, marka, model):
        self.marka = marka
        self.model = model
```

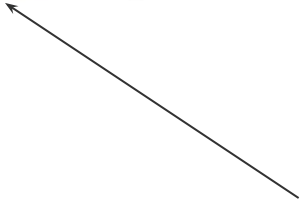
specjalna metoda w Python ma 2 podkreślenia double-under-score in. **dunder**

atrybuty każdej instancji obiektu Samochod

Konstuktor wykonuje **inicjalizację** obiektu

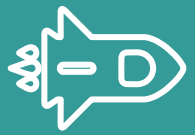
DEFINIOWANIE METOD

```
def accelerate(self, value):
    self.speed += value
```



specjalny parametr automatycznie wypełniana instancją klasy

Przykłady



Pliki:

- `czesc_1/14_obiekty.py`

PRZERWA 15 MINUT

1. Aplikacja okienkowa

TKinter

- Biblioteka umożliwiająca tworzenie interfejsu graficznego dla Pythona, oparta na bibliotece Tk
- *De facto* standard tworzenia GUI w Pythonie
- Klasy “widgetów” reprezentujące elementy interfejsu graficznego (okno, przycisk, napis)

TKinter

```
from tkinter import *

root = Tk()                                  # Stwórz bazowe okno (root)
w = Label(root, text="Hello, world!")        # Stwórz napis “Hello, world!”
w.pack()                                     # Umieść napis na oknie
root.mainloop()                              # Uruchom pętle zdarzeń
```

Źródło:
https://en.wikipedia.org/wiki/Tkinter

Aplikacja okienkowa

- Prosta aplikacja oparta na TKinter:

  Wąż idzie do smakołyka

- Oddzielenie logiki od kodu wyświetlającego
- W pełni obiektowy kod

Pliki:

- `czesc_2/waz.py    # logika aplikacji`
- `czesc_2/main.py   # warstwa prezentacji (aplikacja)`

Thanks!

bartlomiej@biernacki.me

github.com/pax0r