Programowanie w Pythonie

Łukasz Mioduszewski, UKSW 2022

Plan wykładu

- Poznanie środowiska i podstawowych komend
- Funkcje i struktura kodu
- Podstawy programowania obiektowego w Pythonie
- Obliczenia numeryczne - biblioteka numpy
- Praca z obrazem - biblioteki matplotlib i opencv
- Inne biblioteki…

Zasady oceniania

- Ćwiczenia z zadaniami wysyłanymi na Moodle:
- https://e.uksw.edu.pl/course/view.php?id=32351
- Ostatnie ćwiczenia w formie „egzaminu”
  - Średnia ocen z ćwiczeń daje ocenę końcową

> 50 %	3
> 60 %	3+
> 70%	4
> 80%	4+
> 90%	5

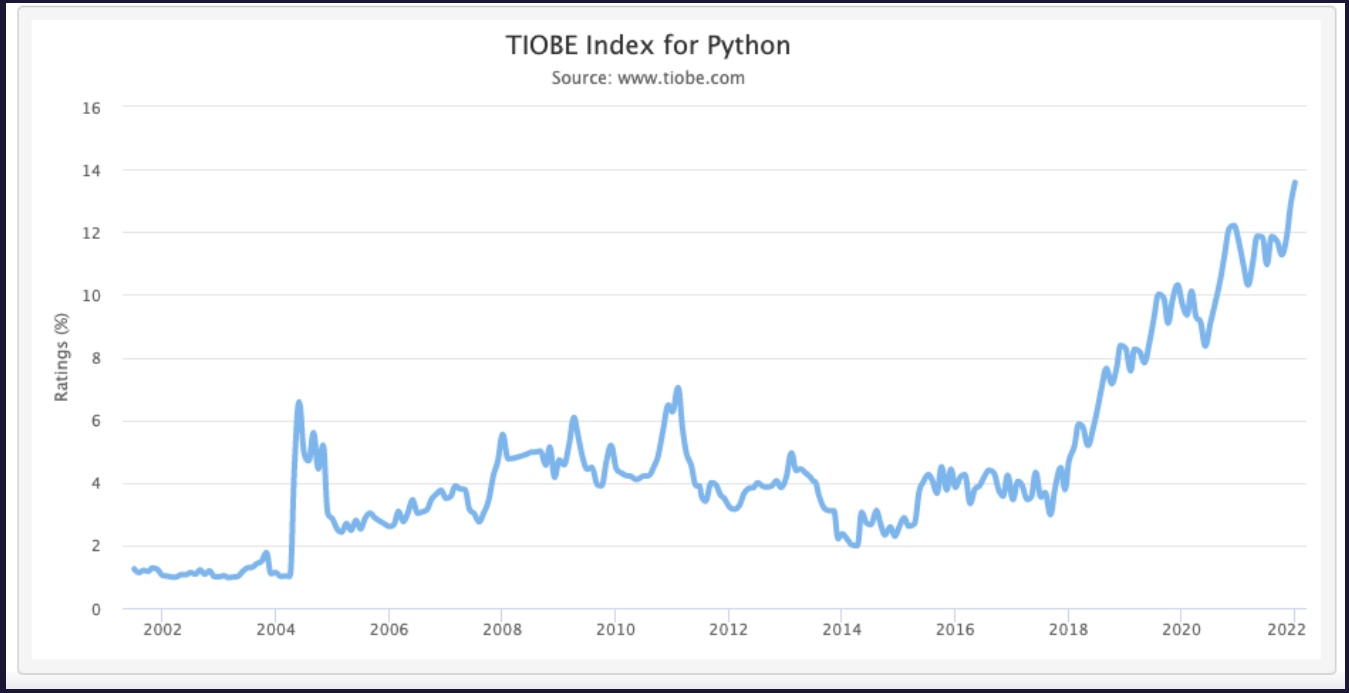
Python – czemu się go uczyć?

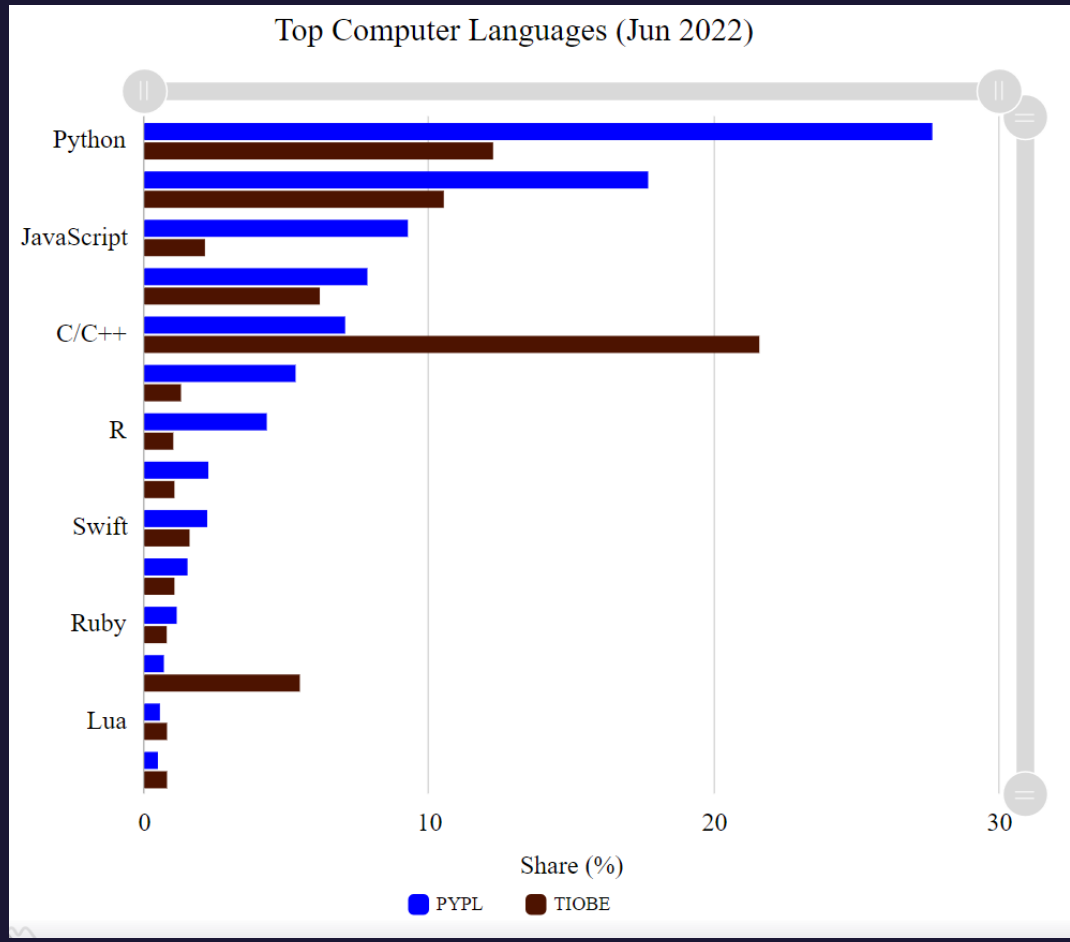
Najbardziej pożądana technologia wg StackOverflow

Najbardziej popularny w Data Science i Machine Learning

Popularność wciąż rośnie

Python – czemu się go uczyć?





Python

- Nazwa pochodzi od Monty Pythona
- Guido van Rossum, 1989: „I was looking for a “hobby” programming project that would keep me occupied during the week around Christmas. My office would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately.

Python - cechy

- Interaktywny i interpretowany
- Wysokopoziomowy
- Obiektowy
- Składnia wymusza czytelność (wcięcia)
- Mnóstwo bibliotek które już robią to co chcesz
- Przenośność (kod będzie działał wszędzie, o ile wersje Pythona i bibliotek będą te same…)
- Możliwość rozszerzenia o moduły z C++, Fortrana itd.

Python - linki

http://www.python.org

http://docs.python.org Dokumentacja

http://docs.python.org/tutorial/ Tutorial

http://wiki.python.org/moin/PolishLanguage Wiki po polsku

Python – skąd wziąć

- https://www.anaconda.com/products/distribution
  - Popularna wersja która ma już wiele bibliotek w standardzie
- https://trinket.io/python3
  - Interpreter w przeglądarce
- Na Linuxie jest od razu (w większości dystrybucji)

Książki o Pythonie

- “A Byte of Python” by C.H.Swaroop; darmowy ebook dla początkujących
- “Dive into Python”; darmowy ebook dla zaawansowanych
- http://aspn.activestate.com/ASPN/Python/Cookbook
  - Zestaw przykładów i wskazówek
- W praktyce jak czegoś nie wiemy to Google -> StackOverflow

Odpalamy Pythona

- Interpreter Pythona to program, który możemy uruchomić w konsoli i wpisywać mu polecenia po kolei, np. ‚print(„Hello World”)’ albo ‚2+2’
- Podobny jest Ipython Notebook, ale bardziej interaktywny niż konsola
- Możemy też napisać skrypt (plik tekstowy) i uruchomić go tak:
  - python skrypt.py
- Są też IDE (Integrated Development Environments):
  - http://wiki.python.org/moin/IntegratedDevelopmentEnvironments

Słowa w Pythonie

- Gdy deklarujemy zmienne, funkcje itp., zaczynajmy OD LITERY
  (a w specjalnych przypadkach od _ albo __)
- Nie używajmy w nazwach znaków takich jak @, $ czy %
- Wielkość znaków ma znaczenie
- Słowa zarezerwowane: and, exec, not, assert, finally, or, break, for, pass, class, from, print, continue, global, raise,def, if, return, del, import, try, elif, in, while,else, is, with, except, lambda, yield True False

Zdania w Pythonie

- Jedna linia to jedna instrukcja, puste linie są dozwolone i zalecane
- WSZYSTKIE WCIĘCIA TRZEBA KONIECZNIE ROBIĆ JEDNAKOWO (TAB ALBO SPACJE)
- Blok instrukcji zaznaczamy wcięciem (nie nawiasami):

```
condition = True
if condition: # pamiętajmy o dwukropku rozpoczynającym wcięcie
        print("True")
else:         # tutaj tez jest dwukropek
        print("False")
```

Zmienne

- Zmiennych nie trzeba deklarować, tworzą się same podczas przypisania wartości
- Typ jest ustawiany automatycznie (nie trzeba pamiętać o int, long, float, complex)
- Typ zmiennej można sprawdzić poleceniem type:

```
>>> a = 1.
>>> type(a)
<type ’float’>
>>> a = 1
>>> type(a)
<type ’int’>
>>> a = ’1’
>>> type(a)
<type ’str’>
```

Liczby zespolone

- Domyślne w Pythonie, jednostka urojona to j
- Przykłady:

```
>>> print(complex(2))
(2+0j)
>>> print((2+3j)*(5-2j))
(16+11j)
>>> print((2+3j).real)
2.0
>>> z=3-4j
>>> print(z.imag)
-4.0
```

Mutable vs Immutable

- Mutowalne vs niemutowalne? Zmienialne vs niezmienialne?
- „Zawartość” zmiennej może być traktowana jako całość (np. liczba 5 to po prostu liczba 5 i tyle) albo jako coś złożonego (np. lista [1,3,5] ma 3 elementy)
- Zmienne są REFERENCJAMI do ich zawartości, więc wiele zmiennych może wskazywać na tę samą zawartość

Mutable vs Immutable

- Zmienne są REFERENCJAMI do ich zawartości, więc wiele zmiennych może wskazywać na tę samą zawartość. Przykład zmiennej niemutowalnej:

```
>>> x=5
>>> y=x
>>> x=6
>>> print(y)
5
```

Mutable vs Immutable

- Zmienne są REFERENCJAMI do ich zawartości, więc wiele zmiennych może wskazywać na tę samą zawartość. Przykład zmiennej mutowalnej:

```
>>> x=[1,3,5] # elementy listy (kwadratowe nawiasy) oddzielamy przecinkami
>>> y=x
>>> x[2]=6      # listy indeksujemy od zera, wiec trzeci element ma indeks 2
>>> print(y)
[1, 3, 6]
```

Mutable vs Immutable

- Zmienne są REFERENCJAMI do ich zawartości, więc wiele zmiennych może wskazywać na tę samą zawartość. Jeśli tego nie chcemy, używamy copy()

```
>>> x=[1,3,5]
>>> y=x.copy()
>>> x[2]=6
>>> print(y)
[1, 3, 5]
```

Mutable vs Immutable

- Zmienne mutowalne:
  - Listy
  - Słowniki: pary klucz-wartość w nawiasach klamrowych, np. {‚imie’: ‚Jan’, 1: [2,3]}
  - Bardziej skomplikowane obiekty
- Zmienne niemutowalne:
  - Liczby
  - Stringi
  - Tuple: specjalne „listy” w nawiasach okrągłych, np. (1,3,5)

Listy

- Dwukropkiem określamy zakres od-do. Można dodawać, zmieniać i usuwać elementy. Przykład:

```
>>> list = [ ’abcd’, 786 , 2.23, ’Jan’, 70.2]
>>> list[0]
’abcd’
>>> list[1:3]
[786, 2.23]
>>> list[3:]
[’Jan’, 70.2]
>>> list.append(’Ewa’)
[’abcd’, 786, 2.23, ’Jan’, 70.2, ’Ewa’]
```

Tuple

- Dwukropkiem określamy zakres od-do. **Nie można** dodawać, zmieniać i usuwać elementów. Przykład:

```
>>> tuple = ( ’abcd’, 786 , 2.23, ’Jan’, 70.2)
>>> tuple[0]
’abcd’
>>> tuple[1:3]
(786, 2.23)
>>> tuple[3:]
(’Jan’, 70.2)
>>> tuple.append(’Ewa’)
AttributeError: 'tuple' object has no attribute 'append'
```

Słowniki

- Możemy dostać się do elementu albo przez [], albo atrybutem get:

```
>>> my_dict = {'name': 'Jack', 'age': 26}
>>> print(my_dict['name’])
Jack
>>> print(my_dict.get('age’))
26
>>> print(my_dict.get('address’))
None
>>> print(my_dict['address’])
KeyError: 'address'
```

Konwersje

- Zwykle dokonują się automatycznie, np.

```
>>> 2+3.5
5.5
```

- Jeśli chcemy być pewni zawsze możemy napisać np.

  float(x), str(n), tuple(lista)

Operatory

- Działają wszędzie gdzie to ma sens
- Przypisanie: =, +=, *=, /=, ...
- Operatory arytmetyczne: + , - , * , ** (potęgowanie), / , // (dzielenie całkowite), %
- Operatory logiczne: > , < , >= , <= , == , !=
- Operatory logiczne można łączyć: not , and , or
- Operatory na pojedynczych bitach: & , | , ^ , ~

Instrukcja if

- # Najprostsza wersja:
  if conditions:
  	statements # to co wewnątrz if jest wciete
- Jeśli chcemy sprawdzać parę warunków pod rząd, możemy użyć konstrukcji elif:
  if expression1:
  	statement(s)
  elif expression2:
  	statement(s)
  elif expression3:
  	statement(s)
  else:
  	statement(s)

Pętla while

- Wykonuje się w kółko dopóki spełniony jest warunek, np.

```
running= True
while running: # pamiętajmy o dwukropku
        running = statements()
else:                   # tutaj tez
        print("End of while loop")
```

Pętla while

- Przykład: program do wypisywania ciągu Fibonacciego:

```
a, b = 0, 1 # można przypisywać wiele zmiennych naraz
while b < 10 : # wszystkie wyrazy mniejsze od 10
        print(b) # pamietajmy o dwukropku i o wcieciach
        a, b = b, a+b
print(„Koniec„)
```

Pętla for

- Ma strukturę ‚for iterator in iterowanyZbior:’, np.
- # Przyklad 1
  for l in "Python":
  print("current letter:", l)
- # Przyklad 2
  fruits = [„papaya", „maracuya", "mango", "pineapple”]
  for f in fruits:
  print("current fruit :", f)
- # Przyklad 3
  n = 5
  for i in range(n):
  print("current index :", i)

Funkcje

- Funkcja to blok kodu, który możemy wywołać w dowolnym miejscu, np.
  def welcome(): # funkcje definiujemy słowem kluczowym def, na koncu mamy :
  print("Hello World!„) # zawartosc funkcji jest wcieta
  welcome()

Funkcje z argumentami

- Argumentom możemy (ale nie musimy) przypisać domyślne wartości, np.

```
def func(a, b=5, c=10):
        print("a = ", a, "b = ", b, "c = ", c)
        return a+b+c
func(3)            # mozemy podac tylko pierwszy argument, a
func(15, c=14) # jesli chcemy pominac drugi argument i zostawic b=5
func(c=50, a=40)
```

Funkcje z argumentami

- Przykład: wyrazy ciągu Fibonacciego mniejsze od n (domyślnie 3):

```
def fib(n=3):
    a, b = 0, 1
    while b < n:
        print(b)
        a, b = b, a+b
```

Dokumentacja (opisywanie kodu)

- def maximum(x, y):
    """Prints the bigger of two arguments"""
    if x > y:
        print(x,"is bigger”)
    else:
        print(y,"is bigger”)

  maximum(3, 5)
  help(maximum)
  maximum(’ala’,’bob’)

Zmienne lokalne i globalne

```
def f_loc(x):
        print(’x is ’, x) # sprawdzamy ile x wynosi wewnatrz funkcji
        x = 2             # nadajemy zmiennej nowa wartosc
        print(’lokalnie x wynosi ’, x)
x = 50
f_loc(x)
print(’globalnie x wynosi’, x)
```

Zmienne lokalne i globalne

```
def f_loc(x): # to nie zadziała, bo w Pythonie parametry funkcji muszą być lokalne
        global x        # deklaracja ze x jest globalny
        print('x is ', x) # sprawdzamy ile x wynosi wewnatrz funkcji
        x = 2           # nadajemy zmiennej nowa wartosc
        print('lokalnie x wynosi ', x)
x = 50
f_loc(x)
print('globalnie x wynosi', x)
```

Biblioteki (moduły)

- Działają jak obiekty, tzn. dostajemy się do ich metod przy użyciu kropki
- Importujemy je przy pomocy „import”, np.

```
import sys
print('The command line arguments are:')
for i in sys.argv: # atrybut argv zwraca wszystkie argumenty programu
        print(i)
print('\nThe PYTHONPATH is', sys.path, '\n') # wypisujemy gdzie szuka modulow
```

- Powyższy skrypt z argumentami 1,2,3 uruchamiamy tak: python program.py 1 2 3
- Nigdy nie nazywaj programu tak jak istniejący moduł!

Tworzenie własnego modułu

- # Nazwa pliku: mymodule.py
  def hi():
  print("Hi, this is me, your module!„)
  version = "0.1"
- # Nazwa pliku: mymodule_demo.py
  import mymodule as mm # mozna importowac pod inna nazwa
  mm.hi()
  print(’Version’, mm.version)

Wersje Pythona

- Python 2 przestał być wspierany w 2020. roku, od teraz „działa” tylko Python 3
- W Python 2 było np. print „hello”, teraz musi być print(„hello”)
  print jest funkcją, więc jej argumenty powinny być w nawiasie
- W Python 2 stringi mogły zawierać tylko znaki ASCII, w Python 3 już całe UTF8
- Biblioteki z Pythona 2 nie działają w Pythonie 3
- Dzielenie liczb całkowitych w Pythonie 2 zwraca liczbę całkowitą (zaokrągloną w dół), a w Pythonie 3 liczbę zmiennoprzecinkową (chyba że użyjemy //)

Ćwiczenia

- Jeśli wysyłasz jakiś plik na Moodle, pamiętaj:
  - W nazwie pliku powinno być imię i nazwisko (a przynajmniej inicjały)
  - Unikaj rozszerzenia .py (jest czasem blokowane), lepiej zostaw .txt
- Zwykle będzie 5 dni na przesłanie rozwiązania
- Zaliczanie na podstawie rozwiązanych zadań, nie wymagam obecności