Wstęp do programowania

Wykład 12. Python

Plan wykładu

1. Kompilatory a interpretery.
2. Praca z językiem Python.
3. Przykłady

- kolekcje
- pętle
- funkcje
- klasy

Python

Pyton jest interpretowanym (skryptowym) językiem programowania. Oznacza to, że kod źródłowy programu nie jest tłumaczony na język maszynowy i wykonywany bezpośrednio przez procesor (tak jak w C/C++, który jest językiem kompilowanym), tylko jest on wykonywany linia po linii przez dedykowany program zwany interpeterem.

Języki kompilowane:

- wydajność,
- detekcja części błędów na etapie kompilacji
- „niskopoziomy” dostępu do sprzętu

Języki interpretowane:

- przenośność
- brak potrzeby kompilacji
- możliwość pracy interaktywnej

Python

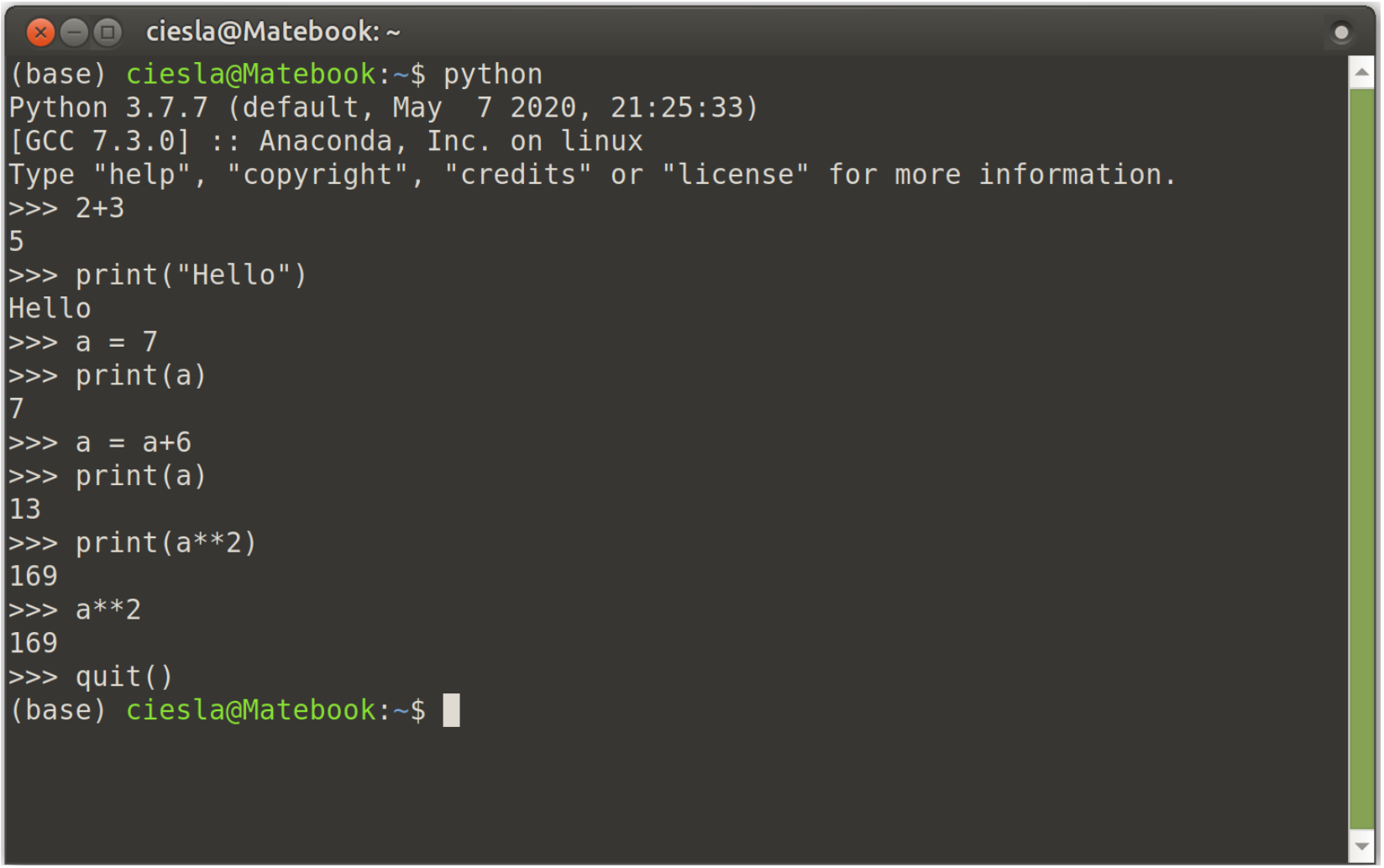
Python obecnie występuje w dwóch wersjach: 2 i 3, przy czym wersja 3 nie zapewnia pełnej wstecznej kompatybilności.

Python jest standardowo dostępny w popularnych dystrybucjach linuxa (najczęściej kompendy python, python2, python3). Wersję dla windows można pobrać ze strony https://www.python.org/downloads/windows/.

Warto rozważyć instalację środowiska programistycznygo zawierającego wiele często używanych bibliotek, np. Anaconda - https://www.anaconda.com/.

Inne popularne rozwiązanie to Jupyter https://jupyter.org/ wykorzystujące przeglądarkę internetową w roli środowiska programistycznego.

Python – tryb interaktywny



Python

Kolejne instrukcje wykonywane przez intepreter można także zapisać w pliku tekstowym (np.. program.py) a następnie wykonać je wszystkie po kolei za pomocą komendy:

```
python program.py
```

Przykłady

Przykładowy program obliczający średnią liczb zapisanych w jednej kolumnie we wsazanym pliku:

```
import math

f = open("plik_z_liczbami","r")
sum = 0.0
sum2 = 0.0
counter = 0
for line in f:
    d = float(line)
    sum += d
    sum2 += d*d
    counter += 1
f.close()
print("srednia = ", sum/counter," sigma = ", math.sqrt(sum2/counter -
(sum*sum)/(counter*counter)))
```

Przykłady

Można też liczby najpierw wczytać do tablicy a potem wykonać obliczenia:

```
import math

f = open("plik_z_liczbami","r")
numbers = []
for line in f:
    d = float(line)
    numbers.append(d)
f.close()

sum = 0.0
sum2 = 0.0
counter = len(numbers)
for d in numbers:
    sum += d
    sum2 += d*d
print("srednia = ", sum/counter," sigma = ", math.sqrt(sum2/counter -
(sum*sum)/(counter*counter)))
```

Przykłady

A można skorzystać z biblioteki **numpy** (lub **pandas**):

```
import numpy as np

f = open("data_n1.0.txt","r")
numbers = []
i = 0;
for line in f:
    d = float(line)
    numbers.append(d)
f.close()

print("srednia = ", np.mean(numbers)," sigma = " np.std(numbers))
```

Przykłady

A można skorzystać z biblioteki **numpy** (lub **pandas**):

```
import numpy as np

f = open("data_n1.0.txt","r")
numbers = []
i = 0;
for line in f:
    d = float(line)
    numbers.append(d)
f.close()

print("srednia = ", np.mean(numbers)," sigma = " np.std(numbers))
```

Kolekcje

W Pythonie dostępne są cztery podstawowe rodzaje kolekcji:

- tablice/listy [] (indeksowane, edytowalne, mogą zawierać duplikaty)
- tuple () (indeksowane, nieedytowalne, mogą zawierać duplikaty)
- zbiory {} (nieindeksowane, edytowalne, bez duplikatów)
- słowniki {} (zbiór par klucz-wartość)

```
mydict = ("key1": "value1", "key2": "value2"}
for key in mydict:
    print(key)
    print(mydict[key])
```

Pętle

W Pythonie mamy standardowy zestaw instrukcji warunkowych i pętli:

```
res = 1
for i in range(1, 100000):
    res *= i
# poniższa linia wypisze!!! wartość 99999!
print(res)


# analogiczny kod z pętlą while
res = 1
i = 1
while i < 100000:
    res *= i
    i += 1
print(res)
```

Funkcje

Funkcje oznaczamy słowem kluczowym **def**.

```
def isEven(x):
    if (x%2)==0:
        return True
    else:
        return False

print(isEven(6))


def divide(nominator, denominator):
    return nominator/denominator

print(divide(20,4))
print(divide(denominator=4, nominator=20))
```

Klasy

W Pythonie można używać klas.

```
class Person:
  # ta metoda to konstruktor
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def sayHello(self):
    print("Hello my name is " + self.name)

p1 = Person("Adam", 23)
p1.sayHello()
# jeśli chemy skasowac object uzywamy del p1
```

Python umożliwia dziedziczenie, natomiast nie udostępnia hermetyzacji (pola i metody private, protected) – wszystko jest publiczne.

Co dalej...

Dodatkowe materiały:

https://www.w3schools.com/python/default.asp

Wykład dr. hab. Andrzeja Kapanowskiego

Dziękuję za uwagę