Podstawy programowania

Wstęp do programowania
w Pythonie

Zakres wykładów

- Podstawy Pythona
- Biblioteki podstawowe
- Biblioteki geoprocesingu
- Geoprocesing w Gdal i PyQGIS
- Analiza i przetwarzanie danych wektorowych
- Analiza i przetwarzanie danych rastrowych
- Biblioteki Pandas, NumPy i SciPy

Literatura

Podstawy języka

- Downey A. B. 2012. Think Python. O'Reilly Media
- Lutz M. 2009. Learning Python, 4th Edition. O'Reilly Media
- Lutz M., Ascher D., 2002, 2010. Python. Wprowadzenie Helion, Gliwice.

Geoprocesing w Pythonie

- Diener M. 2015. Python Geospatial Analysis Cookbook. Packt Publishing.
- Garrard C. 2016. Geoprocessing with Python. Meap Edition.
- Lawhead J. 2015. Learning Geospatial Analysis with Python. Packt Publishing.
- Westra E. 2015. Python Geospatial Analysis Essentials. Packt Publishing.

Źródła internetowe

- http://python.org/doc/
- http://pl.wikibooks.org/wiki/Zanurkuj_w_Pythonie
- http://www.python.rk.edu.pl/w/p/podstawy
- http://geospatialpython.com
- http://www.lfd.uci.edu/~gohlke/pythonlibs/
- https://pcjericks.github.io/py-gdalogr-cookbook/index.html

O Pythonie

- Język Python został stworzony we wczesnych latach 90-tych (1991 roku) przez holenderskiego programistę Guido van Rossum.
- Jest on interpretowanym, obiektowym, wysokopoziomowym językiem, co ułatwia jego testowanie i stosowanie w sposób interaktywny.
- Python to oprogramowanie typu Open-Source zarządzany przez Python Software Foundation, działające na wielu platformach, takich jak: GNU/Linux, Mac OS czy Windows.
- Posiada w pełni dynamiczny system typów i automatyczne zarządzanie pamięcią
- Kod źródłowy napisany w języku Python najpierw kompilowany jest do postaci pośredniej (byte-code), która następnie wykonywana jest przez wirtualną maszynę Python'a (PVM).

Wersje Pythona

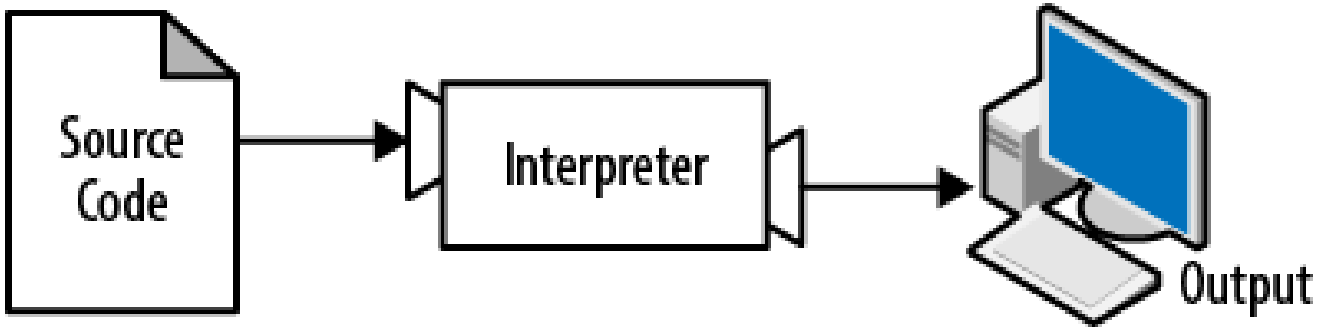
Dwie gałęzie rozwoju Pythona:

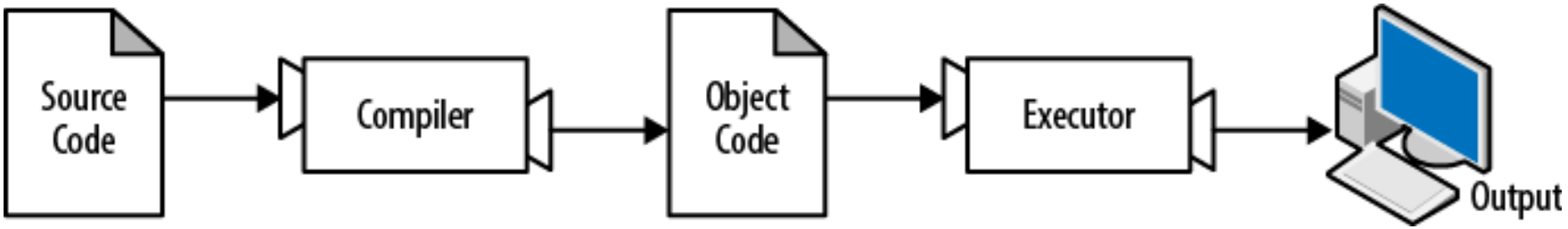
- Python 2.x (obecnie 2.7.14)
- Python 3.x (obecnie 3.7.0)

Interpreter:
https://www.python.org/downloads

Pakiety – dodatkowe biblioteki
https://pypi.python.org/pypi

Interpreter/kompilator



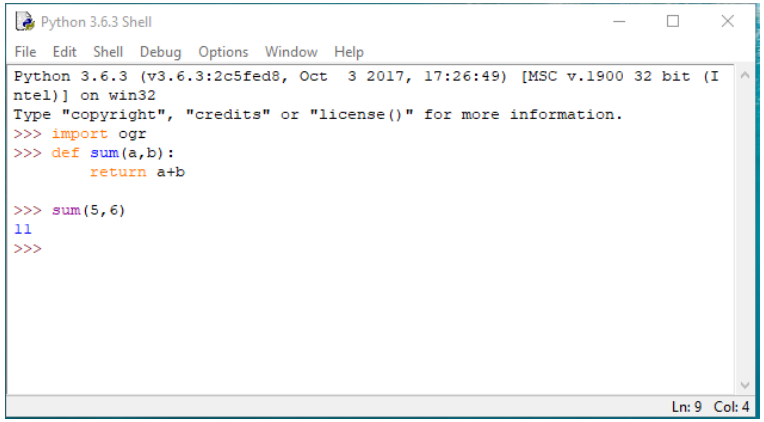


Tryby pracy interpretera

Batch mode - skrypty

python moj_program.py

Interactive mode - terminal



Przykładowy kod

Uwaga: blokowanie kodu przy pomocy wcięć

Kod w Pythonie

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# moj program w Pythonie

def main():

    imie=raw_input("Jak masz na imię ")
    print "Witaj %s"%imie
    d=input("Podaj liczbe ")
    print "Kwadrat tej liczby wynosi %i"%(d*d)
    return 0

if __name__ == '__main__':
    main()
```

Kod w C++

```cpp
#include <iostream>

using namespace std;

int main()
{
    char imie[15];
    int x;
    cout<<"Podaj imie"<<endl;
    cin>>imie;
    cout<<"Witaj"<<imie<<endl;
    cout<<"Podaj liczbe"<<endl;
    cin>>x;
    cout<<"Kwadrat tej liczby wynosi"<<x*x;

    return 0;
}
```

Składnia języka

- Zmienne
- Instrukcje wejścia/wyjścia
- Typy danych
- Instrukcje sterujące
- Funkcje

Liczbowe typy zmiennych

int - liczby całkowite mają rozmiar 32 bitów (max: 2147483647)

long - liczby całkowite większe są oznaczane literą L (np. 9999999999999999L) (tylko w Python 2.*)

float – liczby zmiennoprzecinkowe (3.1415927)

complex -liczby zespolone (3+2.7j)

bool – True/False

Wejście/wyjście Python 2

Pobieranie danych liczbowych do zmiennych:

```
a=input("Podaj wartość")
```

Pobieranie danych tekstowych:

```
imie=raw_input("Podaj nazwę")
```

Wyświetlanie wyników – print jest instrukcją

```
print "Łańcuch znaków"
print  zmienna
print "Liczba całkowita %d i rzeczywista %f"%(a,b)
print "Witaj %s"%(imie)
```

Wejście/wyjście Python 3

Pobieranie danych liczbowych do zmiennych:

```
a=int(input("Podaj wartość"))
b=float(input("Podaj wartość"))
```

Pobieranie danych tekstowych:

```
imie=input("Podaj nazwę")
```

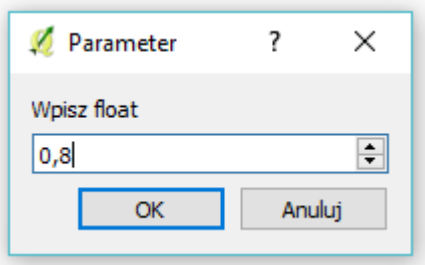
Wyświetlanie wyników – print jest funkcją

```
print ("Łańcuch znaków")
print ( zmienna)
print ("Liczba całkowita %d i rzeczywista %f"%(a,b))
print ("Witaj %s"%(imie))
print("b={1:6d} , a={0:5.2f}".format(a,b))
```

Biblioteka graficzna: PyQt

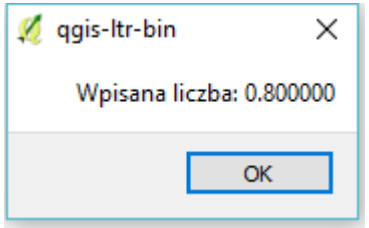
```
from PyQt4.QtGui import * # QGIS 2
from PyQt5.QtGui import * # QGIS 3

text=QInputDialog.getText(None, "Parameter", "Wpisz text")
a=QInputDialog.getInt(None, "Parameter", "Wpisz int")
a=QInputDialog.getDouble(None, "Parameter", "Wpisz float")
```



PyQt

```
msg = QMessageBox()
msg.setText(text[0])
msg.show()
msg.setText("Wpisany tekst: %s"%text[0])
msg.show()
msg.setText("Wpisana liczba: %d"%a[0])
msg.show()
```



Formatowanie liczb

Litera **s** oznacza napis (konwertuje każdy typ danych na tekst), np.:

```
>>> print "%s" % "txt"
```

Litera **c** oznacza pojedynczy znak w kodzie ASCII, np.:

```
>>> print "%c" % "A”
```

Litera **i** oznacza dziesiętną liczbę całkowitą (konwertuje kompatybilny typ danych na liczbę całkowitą), np.:

```
>>> print "%i" % 29
```

Litera **e** oznacza liczbę zmiennopozycyjną w postaci wykładniczej, np.:

```
>>> print "%e" % 1.23
1.230000e+000
```

Litera **f** oznacza liczbę zmiennopozycyjną w postaci ułamka dziesiętnego, np.:

```
>>> print "%f" % 123
123.000000
```

Słowa kluczowe

False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

Złożone typy danych

- Łańcuchy znaków (String)
- Listy
- Krotki
- Zbiory
- Słowniki

Łańcuchy znaków

```
napis=”To jest napis”
len(napis)                  #wynik:13
napis[4]                    #wynik: j
napis[8:12]                 #wynik: napis
napis=napis+napis[7:12]
```

0	1	2	3	4	5	6	7	8	9	10	11	12
T	o		j	e	s	t		n	a	p	i	s

Listy

Listy to pewien rodzaj tablic lub kontenerów

```
>>> lista=["a","b","mpilgrim","z","przyklad"]   #tworzenie listy
>>> lista                                       #zawartość listy
['a', 'b', 'mpilgrim', 'z', 'przyklad']
>>> lista[0]                  #dostęp do elementów listy
'a'
>>> lista[4]
'przyklad'
>>> lista[-1]
'przyklad'
>>> lista[-3]
'mpilgrim'
```

Działania na listach

```
>>> lista[2:3]
['mpilgrim']
>>> lista[2:4]
['mpilgrim', 'z']
>>> lista[:4]
['a', 'b', 'mpilgrim', 'z']
>>> lista.append('nowa')                    #dodawanie elemntów
>>> lista
['a', 'b', 'mpilgrim', 'z', 'przyklad', 'nowa']
>>> lista.insert(3, 'x')                    #wstawianie elemantów
>>> lista
['a', 'b', 'mpilgrim', 'x', 'z', 'przyklad', 'nowa']
```

Działania na listach

```
>>> lista.extend(['jeden','dwa'])
#dodawanie kilku elementów
>>> lista
['a', 'b', 'mpilgrim', 'x', 'z', 'przyklad', 'nowa', 'jeden', 'dwa']
>>> lista.remove('x')     #usuwanie elementów
>>> lista
['a', 'b', 'mpilgrim', 'z', 'przyklad', 'nowa', 'jeden', 'dwa']
>>> len(lista) #długość listy
8
>>> lista.index('z') #sprawdzanie indeksu
3
```

Działania na listach

```
>>> lista=[]
>>> lista.append(1)
>>> lista.append('a')
>>> lista.append('jeden')
>>> lista
[1, 'a', 'jeden']
>>> lista.extend(['dwa', 'trzy'])
>>> lista
[1, 'a', 'jeden', 'dwa', 'trzy']
>>> lista.remove('a')
>>> lista.insert(4, 2)
>>> lista
[1, 'jeden', 'dwa', 'trzy', 2]
```

Metody listy

- **list(s)** - konwertuje sekwencję s np. range(0,9) na listę
- **s.append(x)** - dodaje nowy element x na końcu s
- **s.extend(t)** - dodaje nową listę t na końcu s
- **s.count(x)** - zlicza wystąpienie x w s
- **s.index(x)** - zwraca najmniejszy indeks i, gdzie s[i] == x
- **s.pop([i])** - zwraca i-ty element i usuwa go z listy. Jeżeli nie podamy parametru to usunięty zostanie ostatni element
- **s.remove(x)** - odnajduje x i usuwa go z listy s
- **s.reverse()** - odwraca w miejscu kolejność elementów s
- **s.sort([funkcja])** - Sortuje w miejscu elementy. "funkcja" to funkcja porównawcza

Krotki (niezmienne listy)

Krotka (ang. *tuple*) jest niezmienną listą. Zawartość krotki określamy tylko podczas jej tworzenia. Potem nie możemy już jej zmienić.

```
>>> krotka=("a","b","mpilgrim","z","element")
>>> krotka
('a', 'b', 'mpilgrim', 'z', 'element')
>>> krotka[1]
'b'
>>> krotka[-2]
'z'
>>> krotka[:2]
('a', 'b')
```

Słowniki

Słowniki działają jak tablice asocjacyjne:

```
>>> slownik = {"imie" : "jan", "nazwisko" : "nowak"}
>>> slownik
{'imie': 'jan', 'nazwisko': 'nowak'}
>>> slownik['imie']
'jan‘
#dodanie nowego elementu
>>> slownik['adres']="Warszawa"
#usunięcie elementu
>>> del slownik['imie']
#kasowanie zawartości
>>> d.clear()
```

Zbiory

Zbiory to nieuporządkowane zestawy prostych obiektów. Używamy ich, gdy istotny jest tylko fakt występowania elementu, a nie jego położenie albo liczba powtórzeń.

Zbiory można testować pod kątem występowania danego elementu, sprawdzać czy to jest podzbiór innego zbioru, szukać części wspólnej zbiorów itd.

Przykładowy zbiór

A=set([1,2,3])

```
1  from random import choice
2
3  w=set()
4  while len(w)<6:
5      w.add(choice(range(1,50)))
6  for x in w:
7      print(x)
```

Zamiana złożonych typów

```
>>> li=["a","b","d","a","c","b"]
>>> zb=set(li)  # z listy na zbiór
>>> zb
{'b', 'a', 'c', 'd'}
>>> sl=dict.fromkeys(zb, 0) # indeksy słownika ze zbioru
>>> sl
{'b': 0, 'a': 0, 'c': 0, 'd': 0}
```

Rekrody (klasy)

```
def main():
    #definicja klasy
    class Student:
        nazwisko=""
        imie=""
        ocena=0.0
    #definicja pustej listy
    studenci=[]
    dec='t'
    i=0
    #pobieranie danych do rekordów
    while dec=='t':
        studenci.append(Student())
        studenci[i].nazwisko=raw_input("Podaj nazwisko")
        studenci[i].imie=raw_input("Podaj imie")
        studenci[i].ocena=input("Podaj ocene")
        i+=1
        dec=raw_input("Czy chczes jeszcze podawac t/n")
    for s in studenci:
        print "%3i. %-14s %-10s %7.1f" % (studenci.index(s)+1, s.nazwisko, s.imie, s.ocena)
    return 0
```

Instrukcje sterujące

- Instrukcja warunkowa
- Instrukcja iteracyjna for
- Instrukcja iteracyjna while

Instrukcja if

```
d=int(input("Podaj liczbę"))

if d%2==0:
    print("liczba parzysta")
else:
    print("Liczba nieparzysta")
```

Jeżeli Prawda

Jeżeli Fałsz

Pętla for

```
for x in range(1,10,1):
    print "%4i" % x
```

range(początek,koniec,krok) – sekwencja

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
li=['pn','wt','sr','cz','pt']
for i in li:
    print i
```

Pętla for i słownik

```
slownik = {'imie': 'Jan', 'nazwisko': 'Kowalski', 'wiek': 12, 'zawód': 'spawacz'}
for klucz, wartosc in slownik.items():
    print(klucz + ': ' + str(wartosc))
```

Instrukcja while

```
1  from random import choice
2
3  w=set()
4  while len(w)<6:
5      w.add(choice(range(1,50)))
6      print(w)
```

Funkcje

Definicja funkcji musi zawierać:

- **nagłówek** funkcji obejmujący
  - **nazwę funkcji**, która pozwoli zidentyfikować funkcję w pozostałej części programu
  - **listę argumentów**, która funkcja otrzymuje na początku działania programu
- **ciało funkcji**, zawierające instrukcje, które zostaną wykonane w momencie wywołania (użycia) funkcja
  - jeżeli funkcja ma zwracać jakiś rezultat, musi zawierać odpowiednią instrukcję **(return)**

Funkcje

```
from random import choice

#definicja funkcji pierwiastek
def pierw(n):
    return n**0.5

def main():

    Wylosowane = set()
    while len(Wylosowane) < 6:
        Wylosowane.add(choice(range(1,50)))
    for x in Wylosowane:
        print "%i = %f"%(x,pierw(x))

    return 0
```

Funkcje i listy

```
from random import choice

def srednia(tab):
    sum=0.0
    for i in range(0,10,1):
        sum=sum+tab[i]
    return sum/10


def main():
    tab=[]
    for i in range(0,10,1):
        tab.append(choice(range(0,100)))

    print "Srednia: %f"%srednia(tab)

    return 0
```

Przykład funkcji

```
def printMax(a, b):
   if a > b:
       print a, 'is maximum'
   elif a == b:
       print a, 'is equal to', b
   else:
       print b, 'is maximum'

>>> printMax(3, 4)
4 is maximum
>>> x = 5
>>> y = 7
>>> printMax(x, y)
7 is maximum
```

Wyrażenie lambda

Python za pomocą pewnych wyrażeń pozwala nam zdefiniować jednolinijkowe mini-funkcje. Te tzw. funkcje lambda są zapożyczone z Lispa i mogą być użyte wszędzie tam, gdzie potrzebna jest funkcja.

Definicja i działanie zwykłej funkcji f i wyrażenia lambda g:

```
>>> def f(x):
...     return x*2
...
>>> f(3)
6
>>> g=lambda x: x*2
>>> g(3)
6
```

Komunikacja z plikami tekstowymi

```
#Otwarcie pliku do zapisu
   f1 = open("plik1.txt","w")
#zapis do pliku
   f1.write("Pierwsza linia")
   f1.write("\nDruga linia")
#zakończenie zapisu
      f1.close()
#otwarcie do dopisywania
      f1 = open("plik1.txt","a")
#otwarcie do czytania i pisania
   f1 = open("plik1.txt","r+b")
#czytanie treści pliku:
   print f1.read()
```

```
li=['pn','wt','sr','cz','pt']
f1 = open("plik.txt","wb")
for i in li:
    print i
    f1.write("%s\n"%i)
f1.close()
```

Przykład komunikacja z plikami tekstowymi

```
from random import choice

#definicja funkcji pierwiastek
def pierw(n):
    return n**0.5

def main():

    Wylosowane = set()
    while len(Wylosowane) < 6:
        Wylosowane.add(choice(range(1,50)))
    #Otwarcie pliku do zapisu
    f1 = open("plik1.txt","wb")

    for x in Wylosowane:
        #zapis do pliku
        f1.write("%i = %f\n"%(x,pierw(x)))

    #zakończenie zapisu
    f1.close()
    #otwarcie do czytania i pisania
    f1 = open("plik1.txt","r+b")
    #czytanie treści pliku:
    print f1.read()
    f1.close()
    return 0
```

Standardowa biblioteka Pythona

import nazwa_modułu #ładowanie biblioteki

Dokładny opis w:

http://docs.python.org/2/tutorial/stdlib.html

Moduły standardowe:

os, sys – funkcje współpracy z systemem operacyjnym

math – funkcje matematyczne

random – funkcje losujące

datetime – operacje na danych czasowych, kalendarzowych

re – obsługa wyrażeń regularnych

urllib2 – operacje w Internecie

Moduły analiz numerycznych

- **Numpy** – dostarcza dodatkowe struktury danych takie jak tablice, macierze i narzędzia do pracy na nich
- **Scipy** – rozszerza Numpy o dodatkowe funkcje analityczne
- **Matplotlib** – biblioteka rysująca wykresy i grafikę w opraciu o Numpy i Scipy
- **Pandas** – rozszerza wszystkie powyższe moduły do przetwarzana i analizy danych o dodatkowe typy danych takie jak Serie, DataFrame, szeregi czasowe oraz narzędzia do pracy na nich

Biblioteka matplotlib

https://matplotlib.org/api/pyplot_api.html

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 5, 0.1);
y = np.sin(x)
plt.plot(x, y)
plt.show()
```