Przygotowanie do zajęć z laboratorium “Metody numeryczne”

1. Zainstaluj najnowszą wersję Python

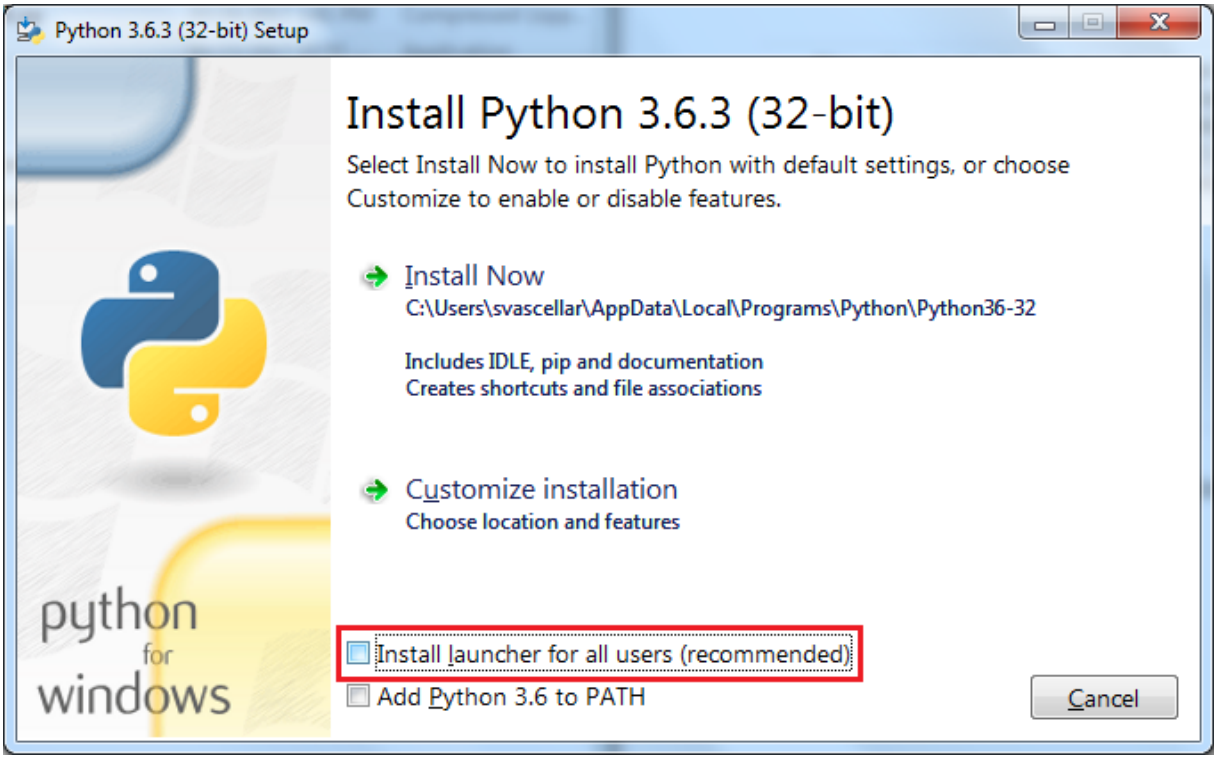
Python jest niezwykle popularnym językiem programowania, który może służyć do tworzenia stron internetowych, gier, oprogramowania naukowego, grafiki i wielu, wielu innych rzeczy.

Początki Pythona sięgają późnych lat 80., a jego głównym założeniem jest czytelność, dzięki której kod jest łatwy do zrozumienia przez ludzi (a nie tylko przez komputery!).

Na początku sprawdź, czy twój komputer działa na 32-bitowej czy 64-bitowej wersji Windowsa, poprzez wciśnięcie kombinacji przycisków Windows + Pause/Break, które otworzą Właściwości systemu i sprawdzenie linii "Typ systemu". Możesz ściągnąć Pythona dla Windowsa ze strony https://www.python.org/downloads/windows/.

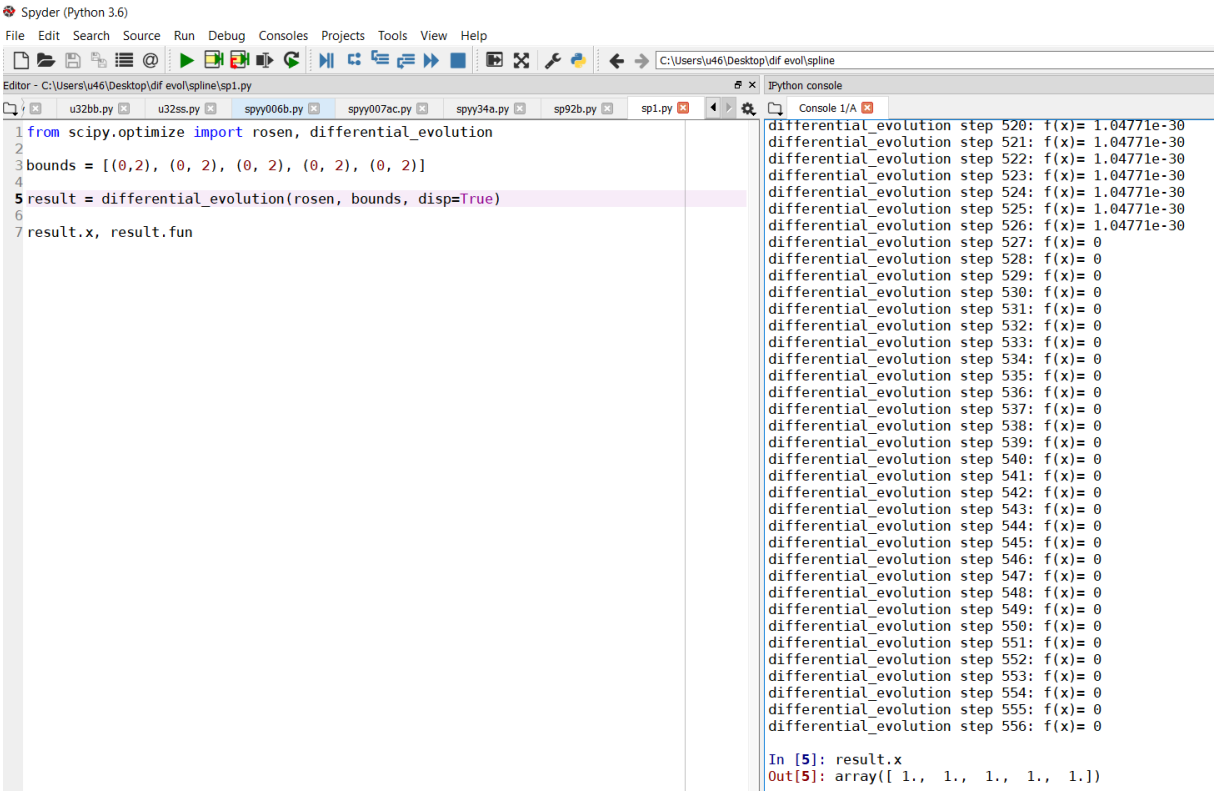
Kliknij w link "Latest Python 3 Release - Python x.x.x". Jeżeli twój komputer pracuje na 64-bitowej wersji Windowsa, ściągnij Windows x86-64 executable installer. W innym wypadku ściągnij Windows x86 executable installer. Po ściągnięciu instalatora, powinnaś go uruchomić (klikając dwukrotnie w niego) i postępować według wyświetlanych instrukcji.

Podczas instalacji zauważysz ekran oznaczony jako "Setup". Upewnij się, że zaznaczyłaś checkbox "Add Python 3.6 to PATH" oraz kliknęłaś "Install Now",



2. Zainstaluj środowisko programistyczne Spyder

Spyder to klasyczne środowisko programistyczne, swoją budową i działaniem mocno wzorowane na RStudio, flagowym narzędziu programistów R. Posiada sporo opcji konfiguracyjnych (np. predefiniowany wygląd przypominający Matlaba itp.), dzięki czemu odnajdą się w nim osoby posiadające doświadczenie z innymi językami programowania. Uzupełnia on dystrybucję Anacondy o narzędzie dla osób wolących tradycyjny sposób programowania.



3. Zaktualizuj biblioteki NumPy i SciPy

Pakiet NumPy

Moduł NumPy jest podstawowym zestawem narzędzi dla języka Python umożliwiającym zaawansowane obliczenia matematyczne, w szczególności do zastosowań naukowych (tzw. obliczenia numeryczne, jak mnożenie i dodawanie macierzy, diagonalizacja czy odwrócenie, całkowanie, rozwiązywanie równań, itd.). Daje on nam do dyspozycji specjalizowane typy danych, operacje i funkcje, których nie ma w typowej instalacji Pythona.

Pakiet SciPy

Pakiet SciPy to zestaw algorytmów numerycznych i przydatnych funkcji, dzięki którym Python może zastąpić dedykowane środowiska do obliczeń numerycznych (Matlab, GNU Octave, Scilab). Pakiet wykorzystuje NumPy do tworzenia wielowymiarowych struktur danych i operacji na nich. Pakiet podzielony jest na podmoduły, z których większość odpowiada tematycznie wybranej dziedzinie metod numerycznych.

Zainstaluj niezbędne dodatkowe biblioteki.

Operacje arytmetyczne

Podstawowe operacje (+, − ,∗,/) działają tak jak można się spodziewać.

In[1]:2+2
Out[1]: 4
Potęgowanie oznaczane jest znakiem ∗∗ , zatem 2 do potęgi trzeciej (2^3) wygląda tak:

In[2]:2**3
Out[2]: 8

Inne operatory arytmetyczne to % — reszta z dzielenia albo modulo: 3 dzielone przez dwa to 1 (bo dwójka mieści się w trójce całkowicie jeden raz) i reszta 1:

In[3]: 3%2
Out[3]: 1

W bardziej skomplikowanych obliczeniach można posługiwać się podstawowym zestawem funkcji (tak zwane funkcje wbudowane):

•abs wartość bezwzględna

•divmod(a,b) zwraca wynik dzielenia całkowitoliczbowego i resztę z dzielenia jako parę liczb

In[4]:divmod(3,2)
Out[4]: (1, 1)

•float konwertuje liczbę do postaci zmiennoprzecinkowej

In[5]: float(3)
Out[5]:3.0

•round(x[,n]) zaokrągla liczbę do n miejsc po przecinku:

In[6]: round(3.1415)
Out[6]:3
In[7]: round(3.1415,1)
Out[7]: 3.1

Użycie innych funkcji matematycznych wymaga specjalnych zabiegów: podpowiedzenia programowi, że będziemy chcieli z nich korzystać:

import math

mówi, że zechcemy używać modułu math zawierającego różne funkcje i stałe matematyczne. Po wydaniu tego polecenia możemy napisać:

```
In[8]: math.cos(math.pi / 3)
Out[8]: 0.5000000000000001
In[9]:math.log(1024, 2)
Out[9]: 10.0 10.0
In[10]:math.pi
 Out[10]:3.1415926535897931
```

Dostępne stają się funkcje trygonometryczne, hiperboliczne, logarytmiczne i wykładnicze, funkcje specjalne oraz stałe: π i e

```
In[10]: math.e
Out[10]: 2.7182818284590451
```

Z całej biblioteki można zaimportować tylko pojedynczą funkcję

```
from decimal import Decimal
```

Obliczenia

W prostych obliczeniach można korzystać z pythona jak z podręcznego kalkulator: wyniki obliczeń podawane będą na bieżąco. Można też wyniki zapamiętywać (coś jak pamięci kalkulatora). Z tym, że poszczególne pamięci możemy nazywać według naszego uznania:

```
 In[1]:r=5
 In[2]:obwod=2*math.pi*r
  In[3]:pole=math.pi*r**2
```

Podanie w linii nazwy pamięci (zmiennej) powoduje wydrukowanie jej wartości:

```
In[4]: r
Out[4]: 5
In[5]: r, pole
Out[5]: (5, 78.539816339744831)
 In[6]: r, pole, obwod
 Out[6]:(5, 78.539816339744831, 31.415926535897931)
```

Bloki instrukcji

Cechą odróżniającą język python od innych języków (Na przykład C) jest sposób wyróżniania bloków instrukcji. W języku C używa się do tego nawiasów klamrowych {}.

W pythonie używa się „wcięć”.

Potrzeba korzystania z bloków instrukcji pojawia się, między innymi:

1.podczas wykonywania instrukcji warunkowych,

2.przy tworzeniu pętli,

3.podczas definiowania funkcji.

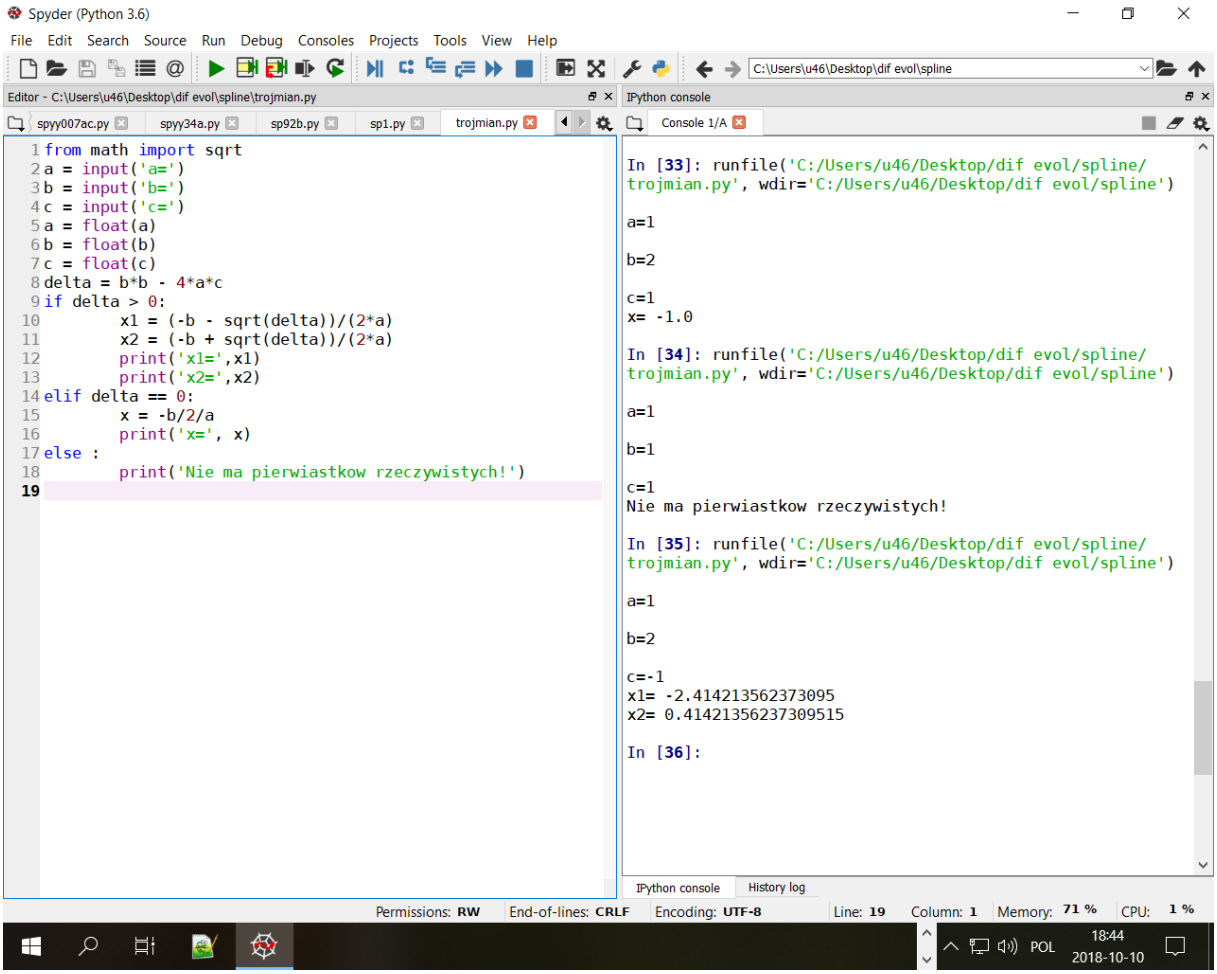
Na przykład:

```
In[7]:for i in range(1,10):
      print(i)
```

```
In[8]:if a > b :
      print( 'A wieksze od B')
else :
      print('B wieksze od A')
```

Poniżej przykład programu rozwiązującego równanie kwadratowe:

```
from math import sqrt
a = input('a=')
b = input('b=')
c = input('c=')
a = float(a)
b = float(b)
c = float(c)
delta = b*b - 4*a*c
if delta > 0:
       x1 = (-b - sqrt(delta))/(2*a)
       x2 = (-b + sqrt(delta))/(2*a)
       print('x1=',x1)
       print('x2=',x2)
elif delta == 0:
       x = -b/2/a
       print('x=', x)
else :
       print('Nie ma pierwiastkow rzeczywistych!')
```



Uwagi:

1.Program można zapisać np. pod nazwą: trojmian.py, a później uruchomić w terminalu wypisując python trojmian.py albo uruchamiamy spyder, otwieramy plik źródłowy File → Open i uruchamiamy Run → Run module albo naciskając klawisz F5.

2.Funkcja float została użyta aby skonwertować tekst (ciąg znaków) do liczby. Informacje wczytywane z terminala funkcją input są typu tekstowego!

Instrukcje warunkowe

Instrukcja warunkowa używana jest podczas programowania do wprowadzenia rozgałęzienia w zależności od wartości jakiegoś parametru czy spełnienia jakiegoś warunku. Typowy przykład konieczności wprowadzenia instrukcji warunkowej to program wyliczania rzeczywistych miejsc zerowych trójmianu kwadratowego. W zależności od wartości parametru Δ albo wyliczamy jeden pierwiastek rzeczywisty, albo dwa, albo musimy stwierdzić, że pierwiastków rzeczywistych nie ma.

Fragment programu wyliczającego pierwiastki przedstawiono w poprzednim rozdziale. (Tam gdzie się kończy wcięcie — kończy się zakres działania odpowiedniego fragmentu warunku.)

Pętle

Pętla to taka konstrukcja programistyczna, która nakazuje wykonanie jakiejś czynności:

•kilka razy

```
for i in range(3,12) :
    print( i)
```

•tak długo aż jakiś warunek zostanie spełniony.

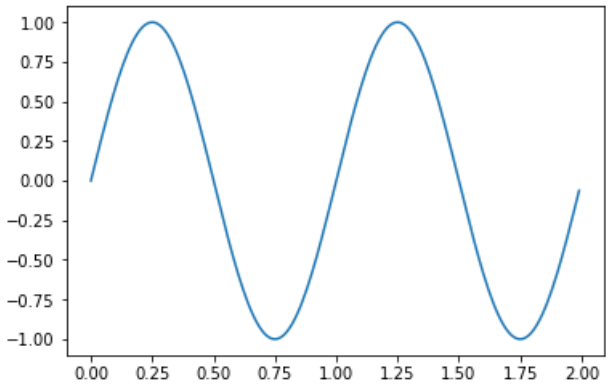
```
i=3
while i < 12 :
    print( i)
    i = i+1
```

W obu powyższych przypadkach efekt powinien być taki:

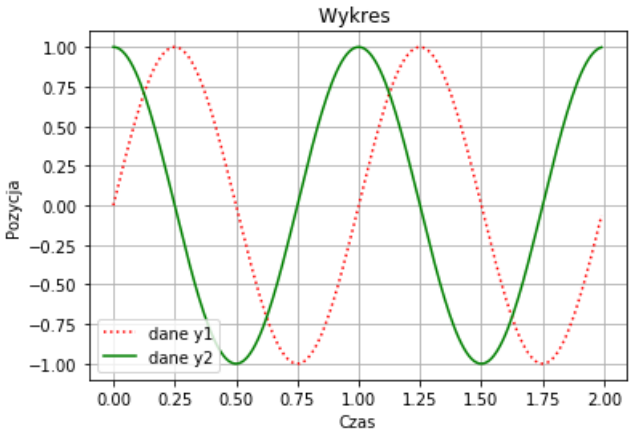
3
4
5
6
7
8
9
10
11

Rysowanie funkcji

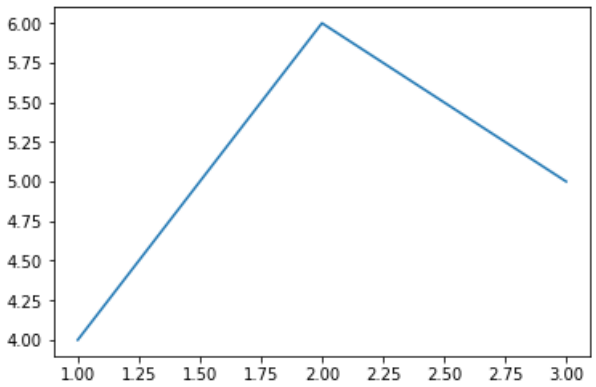
```
In [36]: import numpy as np
    ...: import matplotlib.pyplot as plt
    ...: x = np.arange(0.0, 2.0, 0.01)
    ...: y = np.sin(2.0*np.pi*x)
    ...: plt.plot(x,y)
    ...: plt.show()
    ...:
    ...:
```




```
In [3]: import numpy as np
   ...: import matplotlib.pyplot as plt
   ...: x = np.arange(0.0, 2.0, 0.01)
   ...: y1 = np.sin(2.0*np.pi*x)
   ...: y2 = np.cos(2.0*np.pi*x)
   ...: plt.plot(x,y1,'r:',x,y2,'g')
   ...: plt.legend(('dane y1','dane y2'))
   ...: plt.xlabel('Czas')
   ...: plt.ylabel('Pozycja')
   ...: plt.title('Wykres ')
   ...: plt.grid(True)
   ...: plt.show()
   ...:
   ...:
```



```
In [4]: import matplotlib.pyplot as plt
   ...: x = [1,2,3]
   ...: y = [4,6,5]
   ...: plt.plot(x,y)
   ...: plt.show()
   ...:
   ...:
```



Definiowanie własnych funkcji

In[1]:def dodaj(z1, z2):
        return z1 + z2

tworzy funkcję sumującą oba argumenty (ważne jest wcięcie wszystkich instrukcji)

In[2]:dodaj(1,2)
Out[2]:3

wywołuje odpowiednią funkcję

```
In [1]: def dodaj(z1, z2):
   ...:             return z1 + z2
   ...:
   ...:

In [2]: dodaj(1,2)
Out[2]: 3
```

Opracowanie sprawozdania z laboratorium zawiera opis:

1. Instalacji Pythona 3.X
2. Instalacji środowiska programistycznego Spyder
3. Instalacji bibliotek numerycznych
4. Test działania Pythona w oparciu o omówione konstrukcje programistyczne (należy zaprezentować podobne, ale nie identyczne).