Piotr Szczypiński

Biostatystyka
Ćwiczenia laboratoryjne

część 1

Wprowadzenie do programowania w języku Python

Wprowadzenie do programowania w języku Python

Ćwiczenia laboratoryjne będą prowadzone z wykorzystaniem języka programowania Python. Jest to język interpretowany, dostępny w systemach Linux, OS X oraz Windows. Ćwiczenia będą prowadzone z wykorzystaniem systemu Linux w dystrybucji Kubuntu. Dlatego niektóre informacje zawarte w tej instrukcji będą specyficzne dla tego systemu. Jednak w większość zakresu prezentowanego materiału może być z powodzeniem wykonana również w innych systemach operacyjnych, na których można zainstalować interpreter Python.

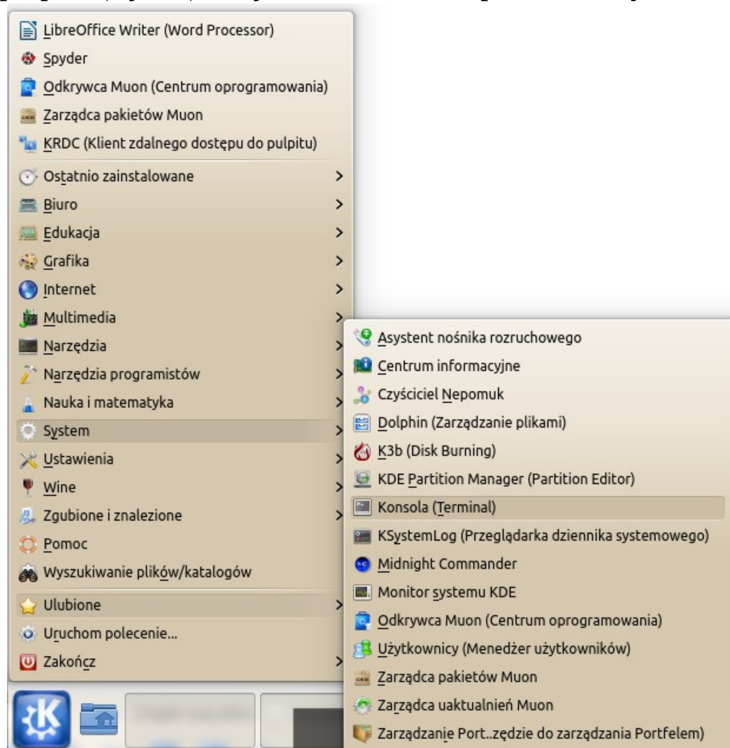
Informacje na temat użytkowania systemu Linux w dystrybucji Kubuntu można znaleźć na stronie: http://www.kubuntu.org/. Dokumentacja interpretera Python dostępne są na: https://www.python.org/

Interpreter Python pozwala programiście korzystać z bogatego zbioru bibliotek zwanych też modułami lub z ang. *extensions*. Wśród tych bibliotek znajdują się trzy, które będą szczególnie przydatne do rozwiązywania problemów z zakresu biostatystyki, są to NumPy, SciPy oraz matplotlib. NumPy udostępnia narzędzia do pracy z funkcjami matematycznymi, macierzami, itp. Biblioteka SciPy zawiera narzędzia do zaawansowanych obliczeń naukowych, w tym analizy statystycznej. Biblioteka matplotlib pozwala wizualizować dane, np. generować wykresy.

Dokumentację bibliotek NumPy, SciPy oraz matplotlib można znaleźć na stronach: http://docs.scipy.org/ i http://matplotlib.org/

Uruchomienie interpretera

W najprostszym przypadku do programowania w Pythonie wystarczy konsola systemowa. W przypadku systemu Windows konsolę uruchamia się poleceniem `cmd.exe`. W dystrybucji Kubuntu konsolę można uruchomić za pomocą menu KDE (*System* > *Konsola*) na dolnym pasku pulpitu (Rys. 1). Aby uruchomić interpreter należy w konsoli wpisać komendę `python` (Rys. 2).



Rys. 1. Uruchamianie konsoli systemowej (terminala).

```
piotr : python – Konsola
Plik   Edycja   Widok   Zakładki   Ustawienia   Pomoc
piotr@piotr-217A:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
piotr : python
```

Rys. 2. Wygląd konsoli z uruchomionym interpreterem Python.

Można teraz sprawdzić działanie interpretera, np. przypisując wartość do zmiennej oraz wykonując proste obliczenia sumowania lub mnożenia.

Przypisz wartość 5 do zmiennej o nazwie x pisząc:

```
>>> x = 5
```

Teraz sprawdź czy zmienna została utworzona i czy zawiera przypisaną wartość:

```
>>> x
```

Interpreter powinien podać wartość zmiennej x:

```
5
```

Teraz pora na proste działanie, np. dodawania:

```
>>> y = x + 7
```

Interpreter właśnie utworzył nową zmienną i przypisał do niej wynik działania. Można sprawdzić co zawiera zmienna y wpisując jej nazwę:

```
>>> y
```

Interpreter poda przypisaną do zmiennej wartość:

```
12
```

W razie potrzeby można zakończyć pracę z interpreterem wpisując instrukcję:

```
>>> exit()
```

Importowanie bibliotek

Spróbujmy teraz obliczyć wartość sinusa zmiennej x. Funkcja sinus ma nazwę sin a argument funkcji podaje się w nawiasie okrągłym po jej nazwie:

```
>>> y = sin(x)
```

Okazuje się, że interpreter nie rozpoznaje nazwy funkcji `sin`:

```
NameError: name 'sin' is not defined
```

Wynika to stąd, że funkcja `sin` jest zawarta w bibliotece NumPy, której to biblioteki jeszcze nie zaimportowaliśmy. Proces importowania polega na załadowaniu przez interpreter Pythona kodów wykonywalnych biblioteki/modułu. Dopiero po ich załadowaniu można korzystać z funkcji zawartych w bibliotece.

Biblioteki dla interpretera python można importować na kilka różnych sposobów. Poszczególne sposoby zostaną przedstawione na przykładzie modułu NumPy.

<u>Sposób pierwszy:</u>

```
>>> import numpy
```

Po zaimportowaniu biblioteki w taki sposób, z funkcji w niej zawartych można korzystać podając

nazwę biblioteki, kropkę oraz nazwę funkcji. Przykładowo dla funkcji sinus składnia instrukcji jest następująca:

```
>>> y = numpy.sin(x)
```

Funkcja sinus została wykonana co można stwierdzić sprawdzając wartość zmiennej y.

Sposób drugi:
Pisanie nazw funkcji poprzedzone długimi nazwami bibliotek jest często niewygodne. Możemy temu zaradzić definiując skrócone nazwy bibliotek w następujący sposób:

```
>>> import numpy as np
```

W tym przypadku zamiast nazwy modułu `numpy` możemy używać jego skróconej wersji np.
Przykładowo:

```
>>> y = np.sin(x)
```

Sposób trzeci:
Jeśli nie chcemy pisać nazw identyfikujących biblioteki przed nazwami funkcji, wówczas importujemy wybraną funkcję z biblioteki w następujący sposób:

```
>>> from numpy import sin
```

Dzięki temu możemy teraz korzystać z funkcji sinus bez podawania nazwy biblioteki, z której została zaimportowana:

```
>>> y = sin(x)
```

Zamiast jednej, wybranej funkcji, możemy za jednym razem zaimportować wszystkie funkcje danej biblioteki. Stosujemy do tego symbol wieloznaczny (*wildcard*) gwiazdki:

```
>>> from numpy import *
```

Możemy teraz korzystać również z innych funkcji zawartych w bibliotece bez konieczności wpisywania nazwy tej biblioteki przed nazwą funkcji, np.:

```
>>> z = cos(x)
>>> z
0.28366218546322625
>>> v = log(24)
>>> v
3.1780538303479458
```

Zadanie 1:
W dokumentacji biblioteki/modułu NumPy odszukaj informacje dotyczące funkcji udostępnianych przez bibliotekę. Sprawdź działanie kilku wybranych przez siebie funkcji.

Sprawdź, czy funkcja może być argumentem innej funkcji. Co oznacza zapis: `log(cos(x))`?

Czy funkcje mogą przyjmować więcej niż jeden argument?

Wektory i macierze NumPy

Wektor (tablicę jednowymiarową) w formacie zgodnym z biblioteką NumPy definiuje się następująco:

```
>>> v = array([2, 3, 6, 1])
>>> v
array([2, 3, 6, 1])
```

Użyliśmy właśnie funkcji array do utworzenia jednowymiarowej tablicy zawierającej cztery wartości.

Macierz (tablicę dwuwymiarową) w formacie zgodnym z biblioteką NumPy definiuje się następująco:

```
>>> v = matrix([[2, 3, 6], [0, 0, 1], [2, 3, 0]])
>>> v
matrix([[2, 3, 6],
        [0, 0, 1],
        [2, 3, 0]])
```

Użyliśmy funkcji matrix do utworzenia dwuwymiarowej tablicy 3×3.

Przyjmijmy, że chcemy teraz utworzyć wektor z wartościami z kolejnymi stu jeden wartościami rozpoczynającymi się od zera i rosnącymi kolejno o 0.5. Ręczne wpisywanie tylu wrtości byłoby pracochłonne i niewygodne. Można do tego użyć funkcji `linspace` należącej do NumPy. Pierwszy argument funkcji określa wartość pierwszego elementu, drugi argument oznacza wartość ostatniego elementu, natomiast trzeci argument funkcji oznacza liczbę elementów wektora:

```
>>> x = linspace(0, 50, 101)
>>> x
array([0. ,   0.5,   1. ,   1.5,   2. ,   2.5,   3. ,   3.5,   4. ,
        4.5,   5. ,   5.5,   6. ,   6.5,   7. ,   7.5,   8. ,   8.5,
        9. ,   9.5,  10. ,  10.5,  11. ,  11.5,  12. ,  12.5,  13. ,
                         ...
       45. ,  45.5,  46. ,  46.5,  47. ,  47.5,  48. ,  48.5,  49. ,
       49.5,  50. ])
```

Zmienna x jest teraz wektorem zawierającym 101 wartości. Możemy teraz obliczyć funkcję, np. sinus, dla wszystkich wartości wektora x, a wyniki zapisać do innego wektora o dowolnej nazwie, np. y:

```
>>> y = sin(x)
>>> y
array([ 0.        ,  0.47942554,  0.84147098,  0.99749499,  0.90929743,
        0.59847214,  0.14112001, -0.35078323, -0.7568025 , -0.97753012,
       -0.95892427, -0.70554033, -0.2794155 ,  0.21511999,  0.6569866 ,
                                   ...
       -0.36730535, -0.76825466, -0.98110844, -0.95375265, -0.69288495,
       -0.26237485])
```

Wykresy matplotlib

Zwróćmy teraz uwagę, że w zmiennej x mamy dostępne argumenty funkcji a w zmiennej y wartości funkcji. Każda para kolejnych punktów z wektora x i y stanowią współrzędne punktów wykresu funkcji sinus. Współrzędne te można wykorzystać do wytworzenia wykresu funkcji. W tym celu potrzebne będą odpowiednie narzędzia modułu matplotlib, które należy zaimportować:

```
>>> from matplotlib.pyplot import *
```

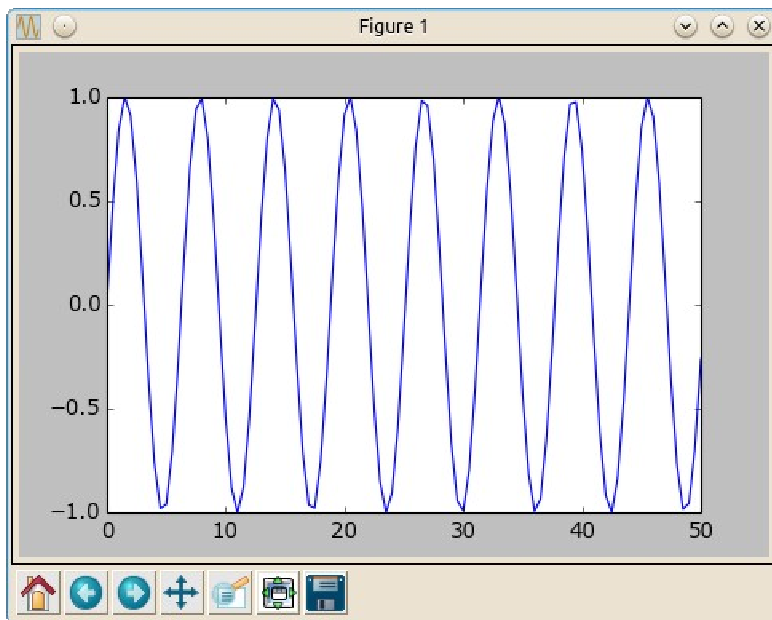
Skorzystamy teraz z funkcji plot. Jej pierwszy argument to wektor współrzędnych w poziomie, drugi to wektor współrzędnych w pionie:

```
>>> plot(x, y)
```

`Wykres może nie być jeszcze widoczny. Aby go wyświetlić należy użyć funkcji:`

```
>>> show()
```

Wynik pokazano na rysunku 3.



Rys. 3. Wykres funkcji uzyskany za pomocą biblioteki matplotlib.

Zadanie 2:
Zamknij teraz okno wykresu. wykonaj następującą sekwencję instrukcji:

```
>>> x = linspace(0, 6, 100)
>>> plot(x, sin(x), 'g-v')
>>> plot(x, cos(x), 'r-o')
>>> show()
```

Sprawdź w dokumentacji biblioteki matplotlib jakie znaczenie ma trzeci argument funkcji `plot` i co oznaczają napisy: `'g-v'`, `'r-o'`, `'g-^'` i `'y +'`?

Zadanie 3:
Wykonaj sekwencję instrukcji:

```
>>> scatter(x, sin(x), 200)
>>> show()
```

W dokumentacji znajdź opis funkcji `scatter`. Czym różni się ta funkcja od funkcji `plot`?

Zadanie 4:
Wykonaj sekwencję instrukcji:

```
>>> subplot(2, 2, 1)
>>> plot(x, cos(x), 'r-o')
>>> subplot(2, 2, 2)
>>> plot(x, sqrt(x), 'g-v')
>>> subplot(2, 2, 3)
>>> scatter(x, cos(x), 200)
>>> subplot(2, 2, 4)
>>> scatter(x, sqrt(x), 50)
>>> show()
```

Znajdź dokumentację funkcji `subplot`, wyjaśnij do czego służy i jakie jest znaczenie jej argumentów. Dokonaj eksperymentów z wykorzystaniem funkcji `subplot` z innymi wartościami argumentów niż podane w powyższym przykładzie.

Zadanie 5:
Znajdź informacje dotyczące tego w jaki sposób wykresy generowane funkcją plot można wzbogacić o opis osi oraz legendę. Przeprowadź testy. Dobrym źródłem informacji jest strona z

samouczkiem: http://matplotlib.org/users/pyplot_tutorial.html.

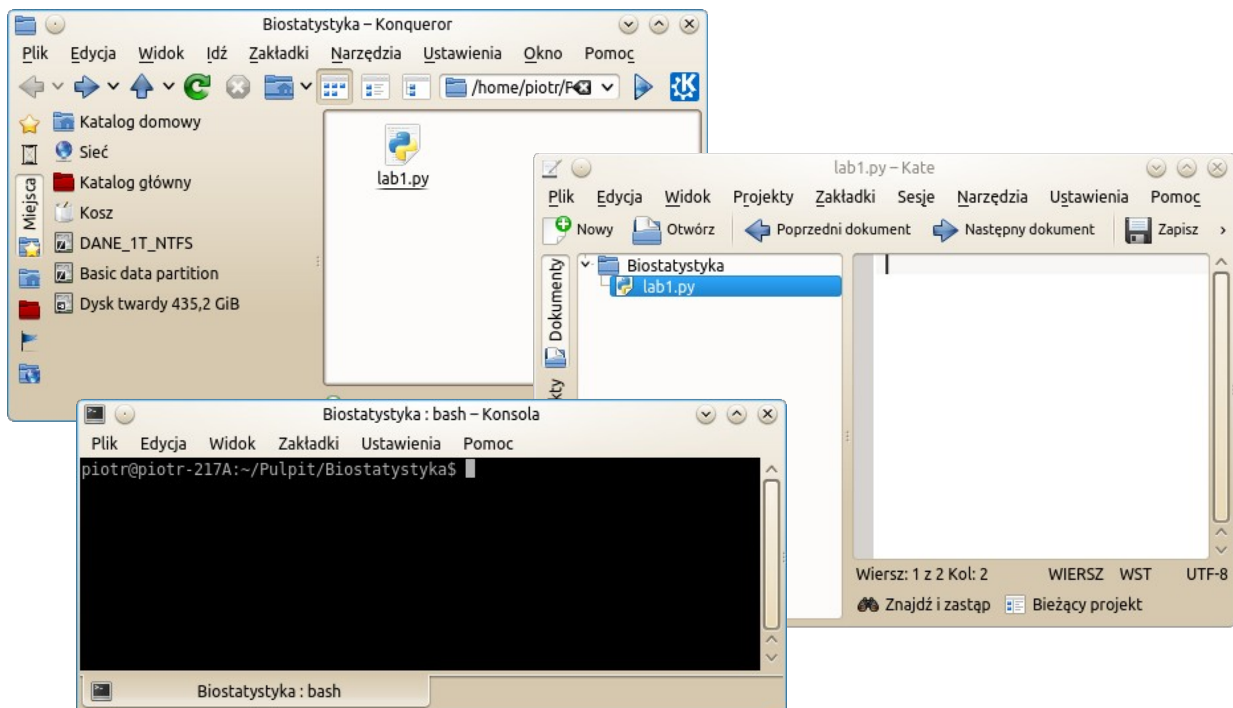
Zadanie 6:
Wykonaj sekwencje instrukcji
(źródło: http://matplotlib.org/mpl_examples/mplot3d/wire3d_demo.py,
http://matplotlib.org/mpl_examples/mplot3d/surface3d_demo.py):

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
ax.plot_surface(X, Y, Z, rstride=10, cstride=10)
plt.show()
```

Dokonaj analizy powyższego kodu: jakie moduły zaimportowano, za co odpowiedzialne są poszczególne funkcje, co przechowują zmienne X, Y i Z?

Skrypty w pythonie

Programowanie w języku Python nie ogranicza się wyłącznie do pisania pojedynczych linii instrukcji w konsoli (terminalu). W języku tym można pisać skrypty, które można później wykonywać bez potrzeby wielokrotnego ręcznego wpisywania sekwencji tych samych instrukcji.



Rys. 4. Okna najprostszych narzędzi wystarczających do pisania i uruchamiania skryptów.

Do pisania skryptów wykorzystamy proste narzędzia takie jak eksplorator systemu plików (konqueror), edytor tekstowy (kate) oraz poznaną już konsolę systemową.

W katalogu domowym lub na pulpicie utwórz nowy katalog o wybranej nazwie (prawy guzik myszy, *Utwórz nowy* > *Katalog*). Otwórz zawartość katalogu za pomocą eksploratora plików (kliknięcie kursorem myszy na ikonie utworzonego katalogu). W otwartym katalogu utwórz nowy plik testowy o nazwie lab1.py (w oknie eksploratora plików, prawy guzik myszy, *Utwórz nowy* >

Plik tekstowy). Otwórz plik tekstowy do edycji w edytorze *kate* (na ikonie pliku lab1.py, prawy guzik myszy, *Otwórz za pomocą* > *Kate*). Przełącz się ponownie na okno eksploratora *konqueror* i wciśnij klawisz F4 – powinno to spowodować otwarcie konsoli systemowej. Na ekranie powinny być widoczne okienka trzech narzędzi tak jak to przedstawiono na rysunku 4.

W edytorze tekstowym wpisz tekst prostego programu:

```
from matplotlib import pyplot as mp
import numpy as np
x = np.linspace(0, 5, 51)
mp.plot(x, np.sin(x), 'r-o')
mp.show()
```

Zapisz program do pliku (guzik *Zapisz* w edytorze kate).

Przejdź do okna konsoli i uruchom skrypt pisząc:

```
python lab1.py
```

Uruchomiony zostanie interpreter python, który z kolei wykona skrypt zapisany w pliku. Po zakończeniu działania skryptu interpreter również zakończy swoje działanie.

W przypadku systemów *unix*, takich jak *linux* możliwe jest uruchomienie skryptu bez informacji na temat tego jaki interpreter jest potrzebny do jego wykonania. Aby było to możliwe, programista skryptu umieszcza w pierwszej linii kodu informację dla systemu o tym jaki program (interpreter) jest wymagany do jego uruchomienia. W przypadku skryptów pisanych w pythonie linijka ta ma postać:

```
#! /usr/bin/python
```

Uzupełnij teraz kod skryptu o podaną powyżej linijkę i zapisz go do pliku. Cały kod powinien teraz mieć postać:

```
#! /usr/bin/python
from matplotlib import pyplot as mp
import numpy as np
x = np.linspace(0, 5, 51)
mp.plot(x, np.sin(x), 'r-o')
mp.show()
```

Następnie ustaw właściwości pliku *lab1.py* wskazując systemowi, że jest to plik z wykonywalnym skryptem. W tym celu przejdź do eksploratora konqueror i ustawiając kursor myszy na ikonie pliku skryptu wciśnij prawy klawisz. Wybierz *Właściwości...* > *Prawa dostępu* i zaznacz pole *Wykonywalny*. Potwierdź wciskając guzik OK.

Teraz skrypt można uruchomić wpisując w konsoli komendę:

```
./lab1.py
```

Zadanie 7:

Wybierz jeden z przykładów generowania wykresów trójwymiarowych ze strony http://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html. Pobierz wybrany skrypt i przetestuj jego działanie. Otwórz plik ze skryptem w edytorze tekstowym i dokonaj jego analizy. Zmodyfikuj kod, zapisz zmodyfikowany plik i uruchom skrypt ponownie.