Python programming and data analysis

Lecture / workshop 1

Robert Szmurło

e-mail: robert.szmurlo@ee.pw.edu.pl

2020L

Where to get?

First of all we will be working with Python 3! Only!

Linux:

- you can use system packgaes manager like `apt get install python3`
- but it is better to use package manager (of your choice):
  Anaconda
  pip

Windows:

- use **Anaconda** (preffered way for newbies) or
- **pip** for avanced users.

How to install? Just ask Google: 'python anaconda install windows'

Environments

- Working with Python (interactive vs scripted)
- Jupyter Notebook (http://jupyter.org/ or https://colab.research.google.com/)
- JetBrains PyCharm (https://www.jetbrains.com/pycharm/)
- Anaconda (https://www.anaconda.com/what-is-anaconda/) or pip (https://pypi.org/project/pip/)
- Visual Studio Code

You can use Jupyter Notebook locally - it wil be much faster, but newbies should forget about yet and they should use Google Colab now!

Python crash course

Plan for today. Basic Python syntax.

- Data types **Numbers** Strings **Printing (% notation, .format, Formatted String Literals (PEP 498 Python >= 3.6, sys.version))** Lists, Dictionaries *** slicing notation (a[start:end:step], start and step can be negative)** Booleans **Tuples** Sets
- Comparison Operators ** if,elif, else Statements
- for Loops
- while Loops
- range()
- list comprehension
- functions
- lambda expressions
- map and filter
- methods

Switch to Jupter Notebook...

Introduction to working with Jupyter Notebook (provided by https://colab.research.google.com)

During the course we shall be programming in Python 3. We can programme in Python language in two approaches:

- using interactive environment
- writing programms (or scripts)

With an interactive approach we usually issue a command and immediatly observe result (which can be printed textual value, a plot or a table). The main interactive environment for Python is IPython. The JupyterNotebook as a wrapping for this interpreter which can be easyli accesse in a web browser.

With a scripted (or programm like) approach we edit longer scripts which can work either in interactive or batch processing modes. For this kind of programming one can use any text editor. However it is encouraged to use some kind of integrated environment which provides some kind of suggestions (intelli sense) and basic type check. There are two commonly used IDEs for Python: JetBrains Pycharm and Microsoft Visual Studio Code.

Lets start with Jupter Notebook. Jupyter Nobeook is a web base interface for a processing environment. Usually this environment is Linux based. Behind the

Let we start coding...

```
!pwd

!uname -a

!lsb_release -a

!pip install pyqt5

!ls -a ..
```

Magic commands

Jupter Notebook interafces the IPython magic commands which provide some additional functionality allowing to manage and control the notebook Python processing environment. On our course we will be mostly using the time execution benchmarking commands (like `%timeit`).

The full list of magic commands can be listed by issuing `%magic` command.

```
# Commented out IPython magic to ensure Python compatibility.
# %magic
```

Basic interactive syntax

We can use IPython as an intelligent calculator…

```
a = 10
b= 24
print(a+b)
c = a+b
print(c)

a = 10.9
a = "abb \"\'bs\'\": " + str(a)
a
```

String formatting

Since we will want to analyse the results of our analysis algorithms, we have good background about string formatting in Python. Because you will probrably be viewing different codes from Internet we shall review all major string formatting approaches (including Python 2.x).

```
a = 10.6
s = 'Mike'
print("%10s has %.1f points." % (s, a) )
s = 'aMike'
print("%10s has %.1f points." % (s, a) )
s = 'aaMike'
print("%10s has %10.1f points." % (s, a+3) )
```

My objective is to print: Mike has 10.6 points.

```
g = "{} has {} points."
print(g.format(s, a) )

print("Something: {} nice {second} {second} {second} {second} third: {}".format("blue", "yellow", s
```

Methods and types discovery

```
type("what it is?")
```

dir - lists all attributes and methods __ - magic methods - standard methods used by common patterns

More string formatting

```
len( "ddddd" )

# dir(str)

print("{named:>30s} has {:.3f} points.".format(a, named=s) )
print("{:^30s} has {:.3f} points.".format(s, a) )

print("{1} has {0} points ({0}).".format(a, s) )

print(f"{s} has {a} points ({a}).");
```

Operating on lists

```
lst = [0,1,2,3,4,4,5, 'Mike', [97,98,99] ]
lst

print(lst[8])
print(lst[8][1])

A = [97, 98, 99]
A.append(1)
print(A)

lst.append( 'kill me' )
lst

print(lst)

help(lst.append)

lst[:5]

lst[:-1]

for i in lst:
  print(i)
  print("----")
```

Even more lists

```
# for (i=0; i<len(lst); i++)
# for i in 0:len(lst)-1
# end

for i in range(0,len(lst)):
print(lst[i])
print("----")

v = list(range(0,10))
print(v)

condition = 1==1
i = 0
while condition:
print(lst[i])
```

Incrementing and comparisons

```
# i = i + 1

i += 1;
condition = i < len(lst)
```

Auditing the **dict** type

```
myd = { 'val1': 23, 'val2': 'Mike'}

print(myd)

myd[34] = 9

print(myd)

print( myd['val2'] )

type(myd)

dir(myd)

help(myd.keys)

myd.keys()

for k in myd.keys():
   print(k)

for k in myd.keys():
   print(f"{k} = {myd[k]}")

list(myd.items())

for k in myd.items():
   print(f"{k[0]} = {k[1]}")
```

More on dict

```
myd.items()

help(myd.items)

for k, v in myd.items():
print(f"{k} = {v}")
```

Tuples - are immutable lists

```
v = ('aaaa', 667, [34, 56])

a,b,c = v

print(a)
print(b)
print(c)

myt = (1,2,3,43)

print(myt[2])

myt[2] = 9
```

Sets, other types and conditional blocks

```
mys = set([0,1,1,1,1,2,1,4,4])
print(mys)

for e in mys:
   print(e)

if 4>3:
   print("Now also true!?!?!")
else:
   print("OK!")

#i = 6
i = 2
if i in mys:
   print("Contains.")
else:
   print("Is not included.")

s = "Ala ma kota"
if "Ala" in s:
   print("Conatins")
```

Functions ...

```
s = "Ala ma kota"
if "ala" in s.lower():
print("Conatins")
else:
print("no")

#help(str)

def myfunc(a,b):
return a+b

c = myfunc(2,3)

print(c)

def myfunc2(a,b):
return a+b, a\*\*2

myfunc2(2,3)

print(myfunc2(2,3)[0])
print(myfunc2(2,3)[1])

n,m = myfunc2(2,3)
print(n)
print(m)
```

String operations

```
sentence = "Robert Szmurlo is giving a lecture on Python programming."
sentence.split()

sentence.upper()

for s in sentence.lower():
   if 'o' in s:
   print(s)

for s in sentence.lower().split():
   if 'o' in s:
   print(s)

def check_o(s):
   if 'o' in s:
      print(s)

list(filter( check_o, sentence.lower().split() ))

def check_o(s):
   if 'o' in s:
      return True

list(filter( check_o, sentence.lower().split() ))
```

Even more...

```
def check_o(s):
   return 'o' in s

list(filter( check_o, sentence.lower().split() ))

list(filter( lambda s: 'o' in s, sentence.lower().split() ))

list(map( lambda s: f"1:{s}", sentence.lower().split() ))

list(map( lambda s: f"{s[0]}:{s[1]}", enumerate(sentence.lower().split()) ))

v = ('a','b','c')
print(list(enumerate(v)))

list(map( lambda s: f"{s[0]}:{s[1]}", enumerate(sentence.lower().split()) ))

list(enumerate(sentence.lower().split()))

help(filter)

# help(str)

lambdas

sentence.split('o')
```

Thank you