Basic tutorial for Python

Jakub Tworzydło

Institute of Theoretical Physics
phone: (022)5532-919, room 5.19
Jakub.Tworzydlo@fuw.edu.pl

1/03/2022 Pasteura, Warsaw

Plan

Plan

Plan

Plan

Features

- easy to learn, very clear syntax, readable code
- interpreter and a scripting language
- oo programing, high-level data structures
- huge standard libraries, special libraries for scientific computing
- open software, good documentation, easy installation

$\Longrightarrow$ idea to use Python for modeling of physical phenomena

Plan

Interactive calculator

First task:

- install Python on your computer
- make sure you can run it in our lab

packages available in all Linux Distro;
Windows users: 70% Python installations

Try from the shell prompt

```
$ python3 -V
```

we will work with 3.4 version (or higher)

Alternatively, if

```
$ python -V
```

gives version > 3.4 it is also fine

Interactive calculator

NUMBERS

```
$ python3

>>> 2+2      # this is a comment
>>> 2/3      # floating point division
>>> 2//3     # integer division

>>> 10**100     # huge number named Googol
                # arbitrary precision integer type

>>> exit()      # to exit interpreter
```

note 1: `$` denotes the shell prompt, `>>>` is Python interactive input

note 2: `$ipython3` invokes nicer interface

VARIABLES

```
>>> h = 20.1        # assigment
>>> w = 3
>>> w * h             # arithmetic
>>> w, h = 5., 6. # multiple assignment

>>> type(w), type(h) # standard data types

>>> t = w, h      # composed variable
>>> t             # known as TUPLE
>>> t[0]
>>> t[1]
```

Numbers and STRINGS

```
>>> s = "12.3"  # is a string
>>> s * 2          # well, it works for strings

>>> float(s) * 2          # this is what we mean
>>> int("2343453656")     # conversion to integer type

>>> str(12.3)             # ... converting to strings
>>> str(2343453656)

# syntax of formatting (three leading digits)
>>> text0 = "Voltage {:.3}".format(2/3)
>>> text1 = "Voltage {:.3} V on lead #{}".format(2/3,1)
```

Complex numbers

```
>>> 1j              # complex unit

>>> (1+2j)*3
>>> (1+2j)/(1+1j)
>>> z = 1.5+.5j

>>> abs(z)          # complex module
>>> z.real
>>> z.imag
```

Module “math”

```
>>> import math  # here we import a module
                 # with lots of useful math

>>> math.factorial(70)
>>> math.factorial(70)/math.factorial(69)

>>> ( math.e**(1j*math.pi) ).imag
>>> ( math.e**(1j*math.pi) ).real

>>> help(math)      # checks the documentation
                    # use q to quit
>>> help(math.tanh)
>>> dir(math)       # lists available commands
```

Strings

```
>>> word = "Help" + "A"    # concatenation
>>> word

>>> "<" + word*5 + ">"

>>> word[0:2]              # accessing substrings
>>> "<" + word[0:2]*5 + ">"

>>> word[:2] + word[3:]
>>> len(word)              # returns length
```

Module for scientific computing “SciPy”

```
>>> import scipy.constants as const
                          # here we import a sub-module
>>> dir(const)
>>> const.h               # Plank's constant
>>> const.h/const.e**2 # quantum of resistance (in Ohms)

>>> const.physical_constants
                          # is a dictionary
>>> const.physical_constants["Planck length"]

-------------------------------------------------------------
>>> import scipy.special as sp
>>> sp.jn(2,0.345)       # Bessel function Jn_2(x)

>>> help(sp)             # ... many are available
```

Plan

Scripts

... set of commands can be stored in a file and executed as a program

Scripts

Set of commands can be collected into a file `*.py` and executed simply by running it `$python3 my_example.py`

Script “Scientific Hello World” illustrates
— how to initialize variables from the command line (for Linux users)
— how to print numbers and text

Create a file `hello.py` containing

```
import sys, math          # import system module

r = float(sys.argv[1]) # reads 1-st argument
                       # from the command line
s = math.sin(r)
text = "Hello, World! "
print(text + "sin(" + str(r) + ")=" + str(s))
```

and run using `$ python3 hello.py 2.3`

Editor

?? use an editor which underlines Python syntax

??? or use an integrated environment e.g. idle

```
$ idle3 &
File -> New Window
print(2+2)
Run -> Module  (or F5)
```

While loop example

Write a simple script as below and run it

```
# Fibonacci sequence:
# sum of two previous elements gives the new one

a, b = 0, 1           # multiple assigment
while b < 10 :        # !!! remember the colon
    print b           # !!! ... and indentation
    a, b = b, a+b

print( "Done!" )
```

Alternatively print results in a single line

```
while b < 1000:
   print(b, end=' ')
```

Lists and for loops

Run and analize (understand) what is printed in the code below:

```
# Create a list consisting of strings
b = ["Mary", "had", "a", "little", "lamb"]

print( b[3] )       # address a single element
print( b[3][1:2] ) #           ... or a letter
print( len(b[3]) )

for i in range(len(b)):     # iterate over index
    print( i, b[i], len(b[i]) )

for x in b:                 # iterate over list
    print( b.index(x), x, len(x) )
```

More list operations

Try to predict what happens in the code, then run it and check

```
# Same list as in the previous example
b = ["Mary", "had", "a", "little", "lamb"]

b.append('!')
print( b )
b.reverse()
print( b )
print( b.pop() )
print( b )
```

Functions

We rewrite our “while loop example” using a function

```
# Define a simple function

def fib(n=3):
    a, b = 0, 1
    while a < n:
      print(a, end=' ')
      a, b = b, a+b

    return( 'Gotowe! \n')


print( fib(2000) )   # function call
print( fib() )       # use optional parameter
```

Local and global variables

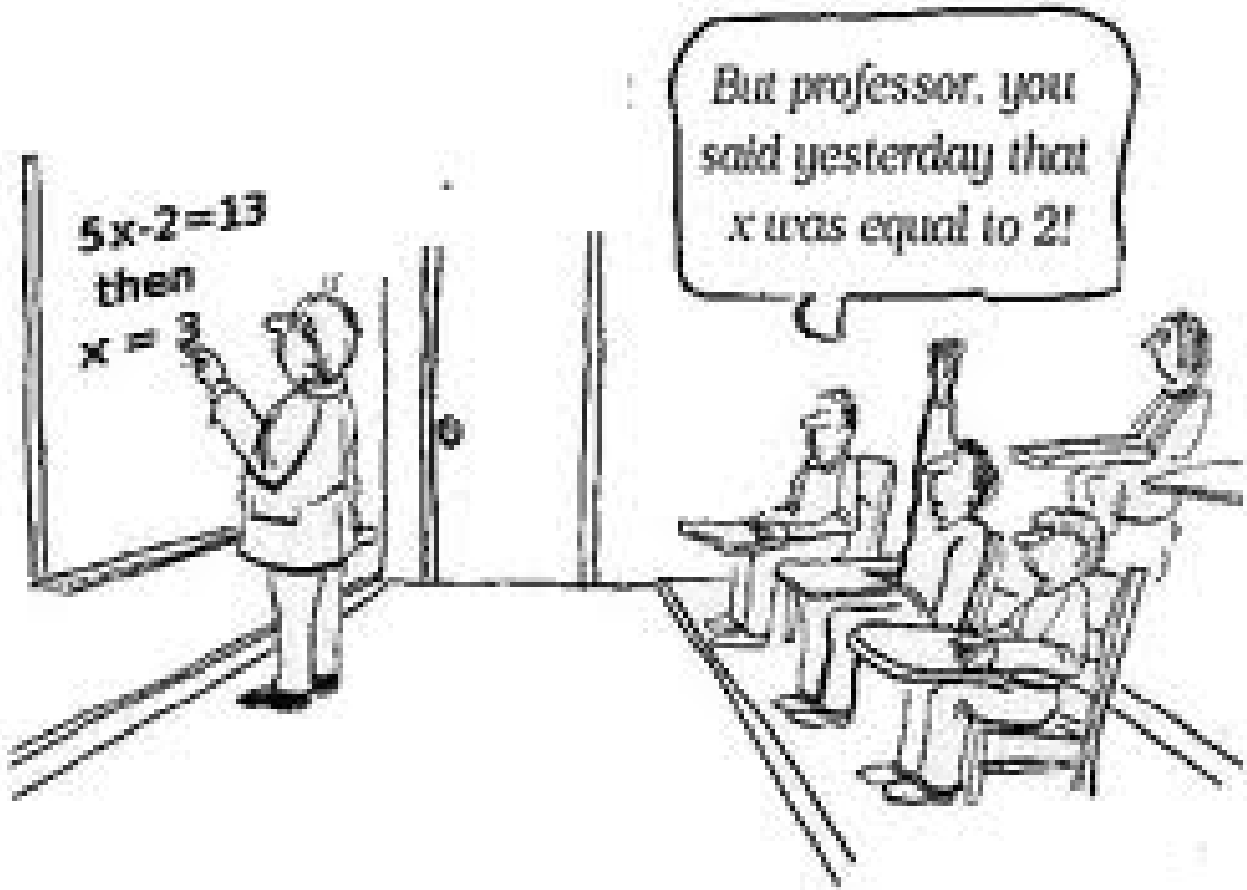
Python understands that local variables do exist only within a function. Run and see what happens.

```
# Filename: func.py

def f_loc(x):
    print( 'x is ', x ) # print to check the value
    x = 2                # a new local variable is created
    print( 'locally x is ', x )

x = 50
f_loc(x)
print( 'x is still', x )
```

Local and global variables



Local and global variables

There is a way to change a global variable within the function

```
# Filename: func.py

def f_glob():
    global x

    print( 'x is', x )
    x = 2
    print( 'x is now changed to ', x )


x = 50
f_glob()
print( 'x in this case is ', x )
```

Integration

```
# SciPy provides an interface to the classical
#                 QUADPACK Fortran library
from scipy import integrate

def my_func(x) :
     return math.sin(x)

res, err = integrate.quad(my_func, 0, math.pi)
print( res, err )

#-----------------------------------------------
# parameters can be included
def your_func(x, a, b):
     return a + b*math.sin(x)

p, q = 0, 1
res, err = integrate.quad(your_func, 0, math.pi,
                          args=(p,q), epsabs=1.0e-6)
```