KEB-45250 Numerical Techniques for Process Modelling
Exercise 1 - Python Tutorial
11.01.2018

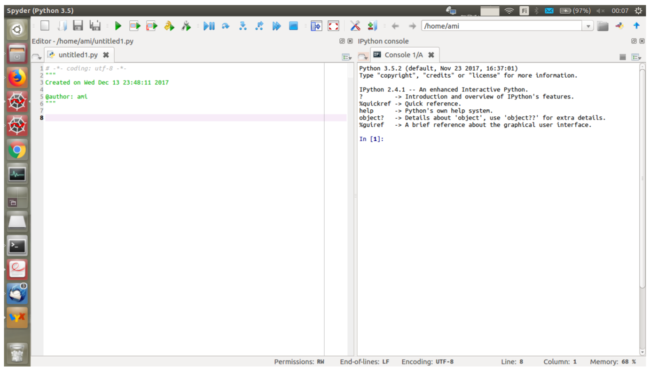
Antti Mikkonen

January 8, 2018

1 Introduction

On this course we use two computational tools: Python and ANSYS. In this exercise session we concentrate on Python. An intensive hand-on course on ANSYS will follow.

The purpose of this exercise session is to familiarize us with Python in the context of scientific calculations. No prior experience with Python is necessary, but programming is not explicitly thought on this course. So if you feel uncertain about your programming and/or Python skills in general we recommend the official Python documentation (https://docs.python.org/3/tutorial/) or the TUT basic programming course "TIE-02100, Introduction to Programming".



```
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 13 23:48:11 2017

@author: ami
"""

```

```
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
%guiref   -> A brief reference about the graphical user interface.

In [1]:
```

Permissions: RW   End-of-lines: LF   Encoding: UTF-8   Line: 8   Column: 1   Memory: 68 %

Figure 1: Spyder3

2 First steps

There are great many ways to access the power of python but we use Spyder3 on this course. Note that we use Python3. Python2 is still widely in use but obsolite. If you want to install the necessary tools on your own computer we recommend the Anaconda package (https://www.anaconda.com/download/#windows) on Windows. For Linux computers use your favorite package manager.

Start up Spyder3 and you'll see something like that on Fig. 1.

On the right you'll see the IPython interpreter. **Start typing simple math in to the IPython**.

```
3+2
4*2
3**3
```

Note that ** is the exponent operator in Python. i.e. 3**3 means 3^3. You'll notice that the IPython acts like a calculator.

Now, create some variables and perform simple math on them. To print a variable on screen use the "print" command.

```
a=3
b=2
c=a*b
print(c)
```

This is convinient for extrimely simple cases but quickly becomes tiresome. So lets **start scripting**. Now type your simple math in the editor window on the left and press **"F5"**. The script you such created will run in the IPython just like if you had writen it there.

Now you have made your first Python script.

3 First toy problem

Now that we have familiarised us with Python scripting lets do something usefull. Consider the pressure drop in a pipe

$$\Delta p = \frac{1}{2}\rho V^2 \frac{L}{d}\mathrm{f} \tag{1}$$

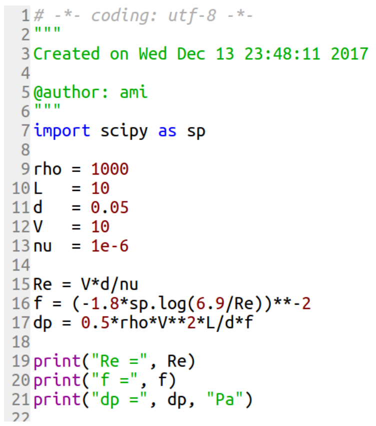
$$\frac{1}{\sqrt{f}} = -1.8\log\left(\frac{6.9}{\mathrm{Re}}\right) \tag{2}$$

$$\mathrm{Re} = \frac{Vd}{\nu} \tag{3}$$

where Δp is pressure drop, ρ is density, V is velocity, L is pipe lenght, d is pipe diameter, f is Darcy friction factor, Reis Reynolds number, and ν is kinematic viscosity. This is a straight forward problem to solve, but note the log-operator in Eq. 2. Log-operator is not available in the standart Python so we need to import a library.

Libraries are imported with "import library_name" command in Python. A library can be given a more simple name with a "as" keyword as shown in Fig. 2. Now we can access the log-function as "sp.log".

Use the same input values as show in Fig. 2 and solve the problem.



```python
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 13 23:48:11 2017

@author: ami
"""
import scipy as sp

rho = 1000
L   = 10
d   = 0.05
V   = 10
nu  = 1e-6

Re = V*d/nu
f = (-1.8*sp.log(6.9/Re))**-2
dp = 0.5*rho*V**2*L/d*f

print("Re =", Re)
print("f =", f)
print("dp =", dp, "Pa")
```

Figure 2: First toy problem

What we just did could have been easily done with a hand held calculator also. Manual calculations are, however, slow and error prone. One you write a script you can easily vary the input parameters. Try playing around.

4 Solving a large number of toy problems

Now lets solve the toy problem in Section 3 for a ten different values of velocity and plot the resulting pressure drop. Let $V = 1, 2, 3, ..., 10 m/s$.

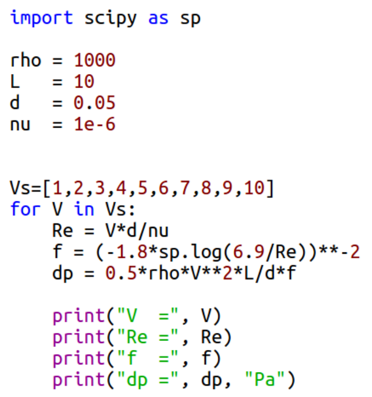
One way to solve this problem is to create a list of the velocity values and the loop through them in a for-loop. Let's start simple by considering the list and loop first and ignore the toy problem for a while. Comment out the previos lines from your sctipt by adding a # symbol in from of all the lines. you can also use the comment-hotkey "Ctrl-1" to comment faster. Commented lines will be ignored by the intepreter.

List are created with square brackets []. Type "Vs=[1,2,3,4,5,6,7,8,9,10]" to create the list and print it with "print(Vs)" command. Now to loop through all the values in the list use the for-loop as

Note that the empty spaces (indent) are part of the Python syntax. Running the code should print all the indivitual V values in Vs, i.e. 1,2,3,4...10.

```
Vs=[1,2,3,4,5,6,7,8,9,10]
for V in Vs:
    print(V)
```

Now uncomment the input values from before (rho, L,...) expect for V and copy-paste the relevant code inside the for-loop, as shown in Fig. 3. Runnning this code now prints the solution for all the velocities.



```
import scipy as sp

rho = 1000
L   = 10
d   = 0.05
nu  = 1e-6


Vs=[1,2,3,4,5,6,7,8,9,10]
for V in Vs:
    Re = V*d/nu
    f = (-1.8*sp.log(6.9/Re))**-2
    dp = 0.5*rho*V**2*L/d*f

    print("V  =", V)
    print("Re =", Re)
    print("f  =", f)
    print("dp =", dp, "Pa")
```

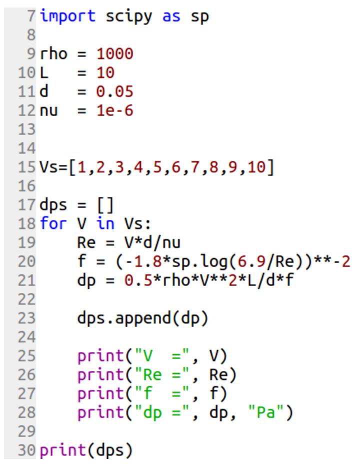
Figure 3: For-loop

Now you have the power to solve a large number of simple problems. Play around.

5 Plotting the results

The print outs in Section 4 are convinient until a certain number of cases but quickly become cumbersome. We now learn how to plot the results in a graphigal manner instead.

First we need to collect the resulting pressure drops (dp) in a new list. First initialize a empty list before the for-loop as “dps=[]” and the append the resulting pressure drop to the new list as “dps.append(dp)”. You can now access the pressure drops afterwards. See Fig. 4.

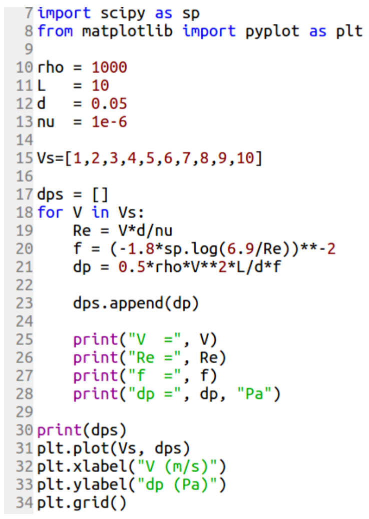


```python
import scipy as sp

rho = 1000
L   = 10
d   = 0.05
nu  = 1e-6


Vs=[1,2,3,4,5,6,7,8,9,10]

dps = []
for V in Vs:
    Re = V*d/nu
    f = (-1.8*sp.log(6.9/Re))**-2
    dp = 0.5*rho*V**2*L/d*f

    dps.append(dp)

    print("V  =", V)
    print("Re =", Re)
    print("f  =", f)
    print("dp =", dp, "Pa")

print(dps)
```

Figure 4: List of pressure drops

The standart tool for scientific plotting in Python is "matplotlib". Import the library and give it a shorter name by adding "from matplotlib import pyplot as plt" in the begining of the script. Matplotlib is a very large library and we only need the "pyplot" part so we only import that.

We can now plot the results as "plt.plot(Vs, dps)". The resulting plot should be visible in the IPython after running the scrip (F5). We can make the plot prettyer by adding labels and grid. See the complete code in Fig. 5.



```python
import scipy as sp
from matplotlib import pyplot as plt

rho = 1000
L   = 10
d   = 0.05
nu  = 1e-6

Vs=[1,2,3,4,5,6,7,8,9,10]

dps = []
for V in Vs:
    Re = V*d/nu
    f = (-1.8*sp.log(6.9/Re))**-2
    dp = 0.5*rho*V**2*L/d*f

    dps.append(dp)

    print("V  =", V)
    print("Re =", Re)
    print("f  =", f)
    print("dp =", dp, "Pa")

print(dps)
plt.plot(Vs, dps)
plt.xlabel("V (m/s)")
plt.ylabel("dp (Pa)")
plt.grid()
```

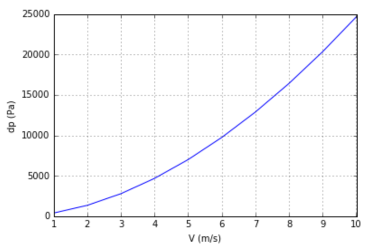


Figure 5: Plotting

6 Scipy Arrays

The lists and loops used in the previous Section work just fine but it is often preferrable to use scipy arrays in stead. This usually results in faster and simpler code. So lets convert our script to scipy array format. You may want to make a back up of your script at this point.

Scipy arrays are basicly lists on steroids. They can do pretty much everything lists can do and a lot more. Lets play around for a while first.

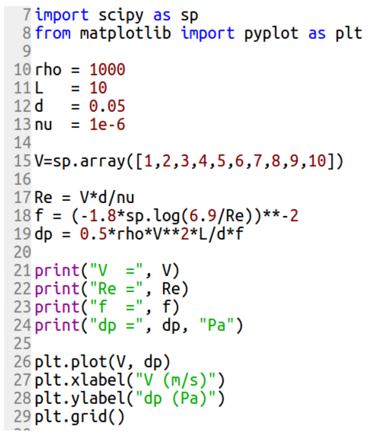
In your IPython intepreted define a couple of scipy arrays of same size, say: “a=sp.array([1,2])” and “b=sp.array([3,4])”. The syntax for the scipy array is a little longer that for the list but quickly pays out.

Now start doing simple math with the newly created a and b. Such as a*b, a**2, sp.log(b),... You’ll notice that all the operations are done element vice. In other words a*b results in two different operations 1*3 and 2*4.

You can also access elements inside your arrays using indexing. For example “a[0]” is the first element in “a” and “b[1]” is the second element in “b”. Therefore “a[0]*b[1]” would give “1*4” with the example values.

Now let’s use scipy arrays in our toy problem. First replace “Vs=[1,2,3,4,5,6,7,8,9,10]” with a equvalent scipy array “Vs=sp.array([1,2,3,4,5,6,7,8,9,10])“. You can rerun the code at this point. Nothing should change.

Now, we can remove the loop from our scipt by using the scipy array instead of single value in our calculations. Also remove the other unnnessesary variables. I also removed the “s” letters from the variable names to make the code prettyer. See complite code in Fig. 6.



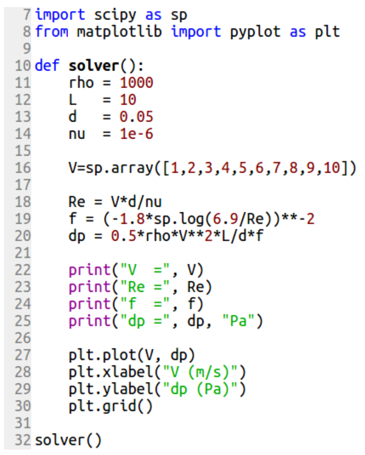
```
import scipy as sp
from matplotlib import pyplot as plt

rho = 1000
L   = 10
d   = 0.05
nu  = 1e-6

V=sp.array([1,2,3,4,5,6,7,8,9,10])

Re = V*d/nu
f = (-1.8*sp.log(6.9/Re))**-2
dp = 0.5*rho*V**2*L/d*f

print("V  =", V)
print("Re =", Re)
print("f  =", f)
print("dp =", dp, "Pa")

plt.plot(V, dp)
plt.xlabel("V (m/s)")
plt.ylabel("dp (Pa)")
plt.grid()
```

Figure 6: Scipy array

7 Defining a function

In order to make our code more structured and reusable it is often useful to define a lot of functions. In this toy problem the function definition is a little artificial but let's do it anyway.

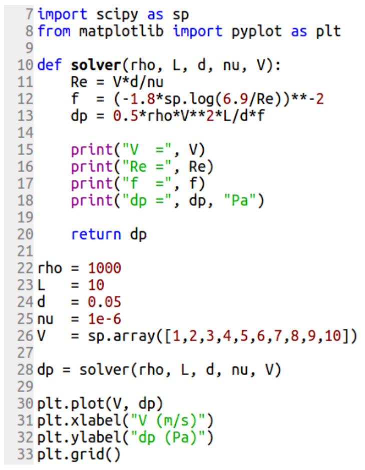
In Python functions are defined with the key word "def". Let's first turn our whole script into a function and call it with no added functionality. See Fig. 7. This should run exactly like before.



```python
import scipy as sp
from matplotlib import pyplot as plt

def solver():
    rho = 1000
    L   = 10
    d   = 0.05
    nu  = 1e-6

    V=sp.array([1,2,3,4,5,6,7,8,9,10])

    Re = V*d/nu
    f = (-1.8*sp.log(6.9/Re))**-2
    dp = 0.5*rho*V**2*L/d*f

    print("V  =", V)
    print("Re =", Re)
    print("f  =", f)
    print("dp =", dp, "Pa")

    plt.plot(V, dp)
    plt.xlabel("V (m/s)")
    plt.ylabel("dp (Pa)")
    plt.grid()

solver()
```

Figure 7: Function definition

Now let's make use the fysical input parameters as paremeters for the function. We also return the pressure drop from the function. See Fig. 8.



```python
import scipy as sp
from matplotlib import pyplot as plt

def solver(rho, L, d, nu, V):
    Re = V*d/nu
    f  = (-1.8*sp.log(6.9/Re))**-2
    dp = 0.5*rho*V**2*L/d*f

    print("V  =", V)
    print("Re =", Re)
    print("f  =", f)
    print("dp =", dp, "Pa")

    return dp

rho = 1000
L   = 10
d   = 0.05
nu  = 1e-6
V   = sp.array([1,2,3,4,5,6,7,8,9,10])

dp = solver(rho, L, d, nu, V)

plt.plot(V, dp)
plt.xlabel("V (m/s)")
plt.ylabel("dp (Pa)")
plt.grid()
```

Figure 8: Function with parameters

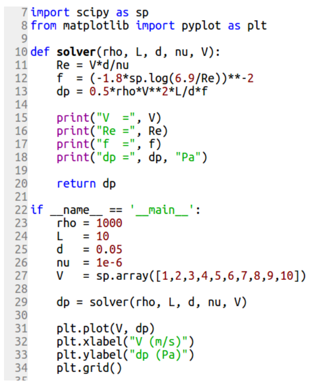
Now we have a solver function that takes all the fysical paramters of the pressure drop as input parameters and return the pressure drop. In a more complicated problem this would likely be a small part of the complite solution and repeted for many different cases.

Now, as a final touch, let's turn our script in to a ready-to-deploy state by adding a "if __name__ == '__main__':" test in the end section.

This line test if the current file is the "main" file running the larger program. The technical term for this is "unit testing" and allows us to test small pieces of code individually without the test code affecting the larger program.

If the unit testing part doesn't make sense to you, never mind. Just believe it's a good programming practice and the reasons will become obvious with experience. Or search internet for "unit test".

The complite code below in Fig. 9.



```python
import scipy as sp
from matplotlib import pyplot as plt

def solver(rho, L, d, nu, V):
    Re = V*d/nu
    f  = (-1.8*sp.log(6.9/Re))**-2
    dp = 0.5*rho*V**2*L/d*f

    print("V  =", V)
    print("Re =", Re)
    print("f  =", f)
    print("dp =", dp, "Pa")

    return dp

if __name__ == '__main__':
    rho = 1000
    L   = 10
    d   = 0.05
    nu  = 1e-6
    V   = sp.array([1,2,3,4,5,6,7,8,9,10])

    dp = solver(rho, L, d, nu, V)

    plt.plot(V, dp)
    plt.xlabel("V (m/s)")
    plt.ylabel("dp (Pa)")
    plt.grid()
```

Figure 9: Unit test