Computation in Python/JUPYTER
Introduction

Yadav, Prabhas. K.

prabhaskumar.yadav@htw-dresden.de

13 Oct. 2022

Today's Contents

What is Python?

Python is an open-source interpreted, high-level, general-purpose programming language. This means:

1. **Interpreted/high-level language**: This makes us avoid the nuances of fundamental coding as done by computer programmers/engineers.
2. **General purpose programming language and Open-source ecosystem**: This means it is extensible. Already over 200,000 **Python** packages are available (check here). Also, it means **FREE** of cost.
3. **For Hydraulics Modelling**: We can use **Python** packages such as `Numpy` (for numerical computing), `Scipy` (for scientific computing), `Sympy` (for symbolic computing), `Matplotlib` (for plotting) etc. for our computing and modelling in the course.

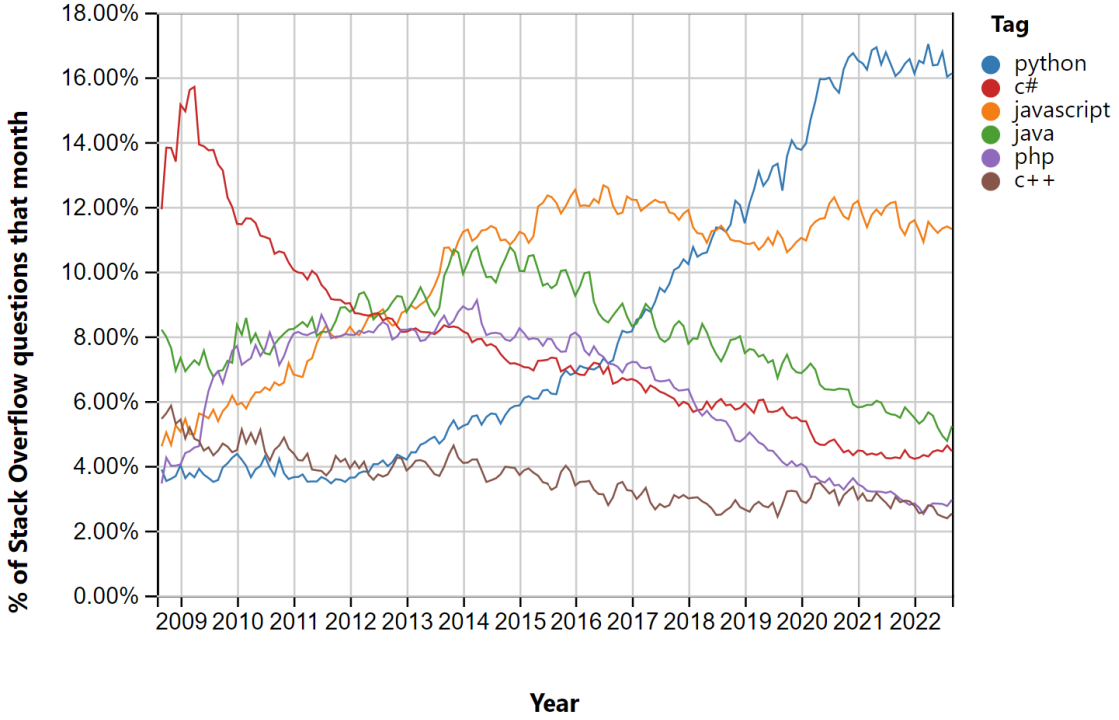
A Bit of Python History

- Guido van Rossum began developing **Python** in 1980 at Centrum Wiskunde & Informatica (CWI), the Netherlands. Its implementation (**Python** v.1) was released in 1994.
- **Python** 2.0, released in 2000 became one of the most used general purpose programming language. **Python** 2.0 is now being replaced by **Python** 3.0 (from 2020).
- **Python** 3.0 will be used in our class. It is **not** 100% compatible with earlier versions of **Python** .
- **Python** name comes from the British comedy group Monty Python (Van Rossum enjoyed their show). The official Python documentation (check here) also contains various references to Monty Python routines.

Why Use Python?

- **Python** is a **common tool** among engineers, experts and researchers at universities and industry.
- **Python** is s**ystem independent**, therefore it is highly portable. Beside, it is a versatile (multi-purpose) language.
- **Python** is incredibly flexible and can be adapted to specific local needs using enormous number of **PACKAGES**. Beside, it can easily interface with other languages
- **Python** is under incredibly active development, improving greatly, and supported **wildly** by both professional and academic developers

Popularity of Python



1. **Python** has become a mainstream computing language.
2. Details on plot are here.
3. This all means - it is good to learn to code in **Python**

Basic of **Python** language

Very basics of **Python** before starting **JUPYTER** Notebook

- **Comments symbol**: with #
- **End statement**: with semicolon (;) or line-end
- **Multi-line statement**: with continuation character (\)
- **Multiple statement**: separate each with **;**
- **Code Indentation**: is required (4 spaces for each)
- **String quotes**: use either " " or ' '
- **Naming**: is case sensitive $b \neq B$
- **Variable Assignment**: use =, e.g. `x=5`
- **Multiple Assignment**: allowed e.g, `a = b = c = 1`
  or `a,b,c = 1,7,10`
- **Help**: use `help()` and **get the cheatsheet from here**

Basic of **Python** language

Data types in **Python**

- **Numbers**: numbers of any types: `int (e.g., 10)`, `long (e.g., 5192L )`, `float (e.g., 13.4)`, `complex (e.g., 2.13j)`
- **String**: with quotation mark, e.g., `"john"`
- **List**: are modifiable placed in **[ ]** separated by comma (**,**). e.g., `[1, 3, "apple", 2, "jo"]`
- **Tuple**: are non- modifiable placed in **( )** separated by comma (**,**). e.g., `(11, 3, "ape", 2, "job")`
- **Dictionary**: For table-like data. Placed in {} separated by comma (**,**), e.g., `{'name': 'john', 'code':6734, 'dept': 'sales'}`

Basic of **Python** language

Basic **operators** in **Python**

Arithmetic Operations

Sym.	**Operation**	**e.g.,**
+/-	Add/Subs.	a+b
*	Multiply	a*b
/	Divide	a/b
**	exponent	a**2 = a^2
%	Modulus	Reminder a/b

Bitwise/Comparison

sym.	**Operation**	**e.g.,**
&	AND	a & b
|	OR	a|b
==	equal to	a==b
!=	not equal to	a!=b
>/<	greater/less	a>b

Refer to **Python** documentation for complete description. **Python** documentation is very extensive and can be obtained from **here**

Basic of **Python** language

A FUNCTION in **Python**

A **function** in a programming provide an ability to develop a reusable code-block with an option of several operations. So, a **function** have a (or a set) of *input* and provide an (or a set) *output*.

Python Function e.g.

```
1 def func1(a,b):
2
3     c = a+b
4     d = c*a
5
6     return c, d
```

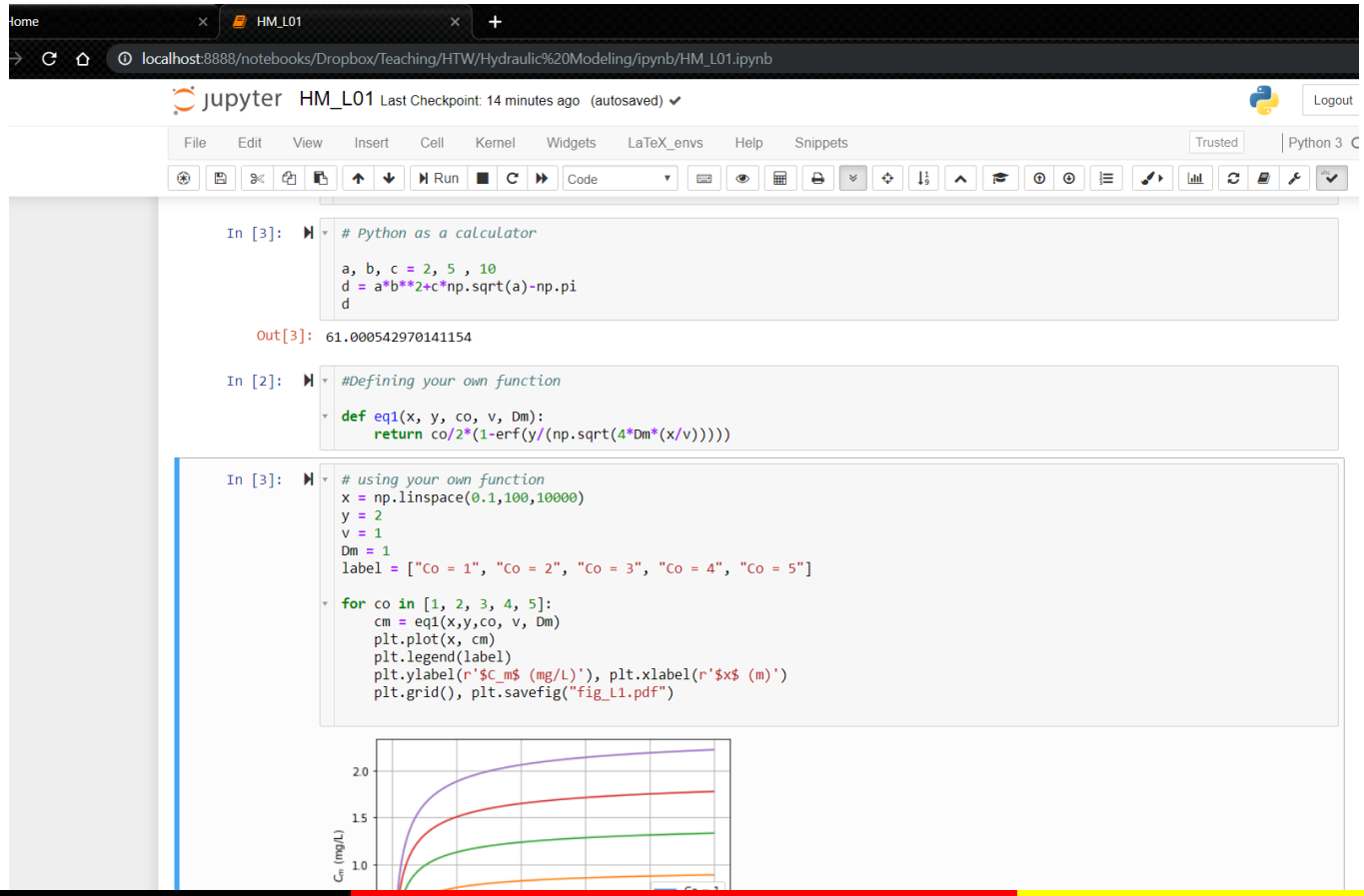
What is it?

def in line-1 begins a `Function` block
func1 in line 1 is user-defined function name
a,b in line 1 are function `input`
c,d in lines 3-4 is function `operation`
return line-6 is function `output` block

Semicolon (:) in line 1 and **Indentation** after line 1 are required. **def, return** are **Python** keywords. There are quite few of them.

The JUPYTER Notebook

We avoid hard-coding and long codes in this course. We use something called **"JUPYTER notebook"**. **Get JUPYTER cheatsheet from here**



More on JUPYTER notebook

JUPYTER – a much better way to work in **Python** that allows on a single display:

- interface for Python and also many other programming language (R, Java, Julia etc.)
- writing and editing of codes (*we will learn this*)
- operation of **Python** and its packages (e.g., Numpy)
- interactively viewing the code output (i.e., results and plots)
- getting **help**

The **JUPYTER** notebook runs in the browser (Chrome, Firefox etc.). So, nothing really required to be installed.

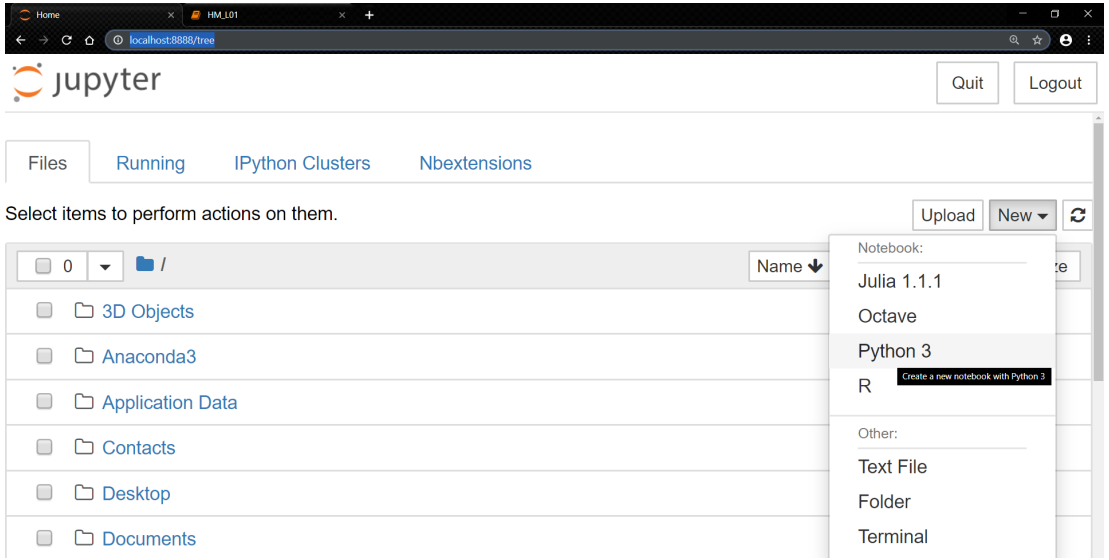
For better computing and learning, you are encouraged to install **JUPYTER** system following the instructions from **here**.

Interacting with JUPYTER notebook

JUPYTER notebook runs in a browser.

The first webpage with **JUPYTER** will have http address, e.g., `http://localhost:8888/tree`. Internet is **not** required.

JUPYTER notebook saves file as `IPYNB` file.



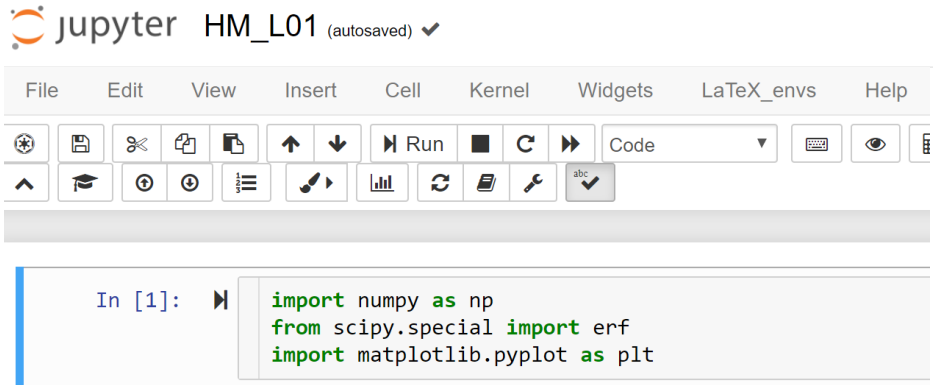
The first-page of **JUPYTER** provides a basic `File-Explorer` of your system - from here it is possible to load available `IPYNB` file or create a new one.

Interacting with JUPYTER notebook

JUPYTER notebook has a block-based interface called **CELL**.

Codes are put in the CELL and executed with **RUN** or using Keyboard buttons **SHIFT+ENTER**

The first `CELL` is usually for importing `PACKAGES` - such as `Numpy`, `Scipy`, `Matplotlib`



Texts in green – (imports, from, as) are **Python** keywords.

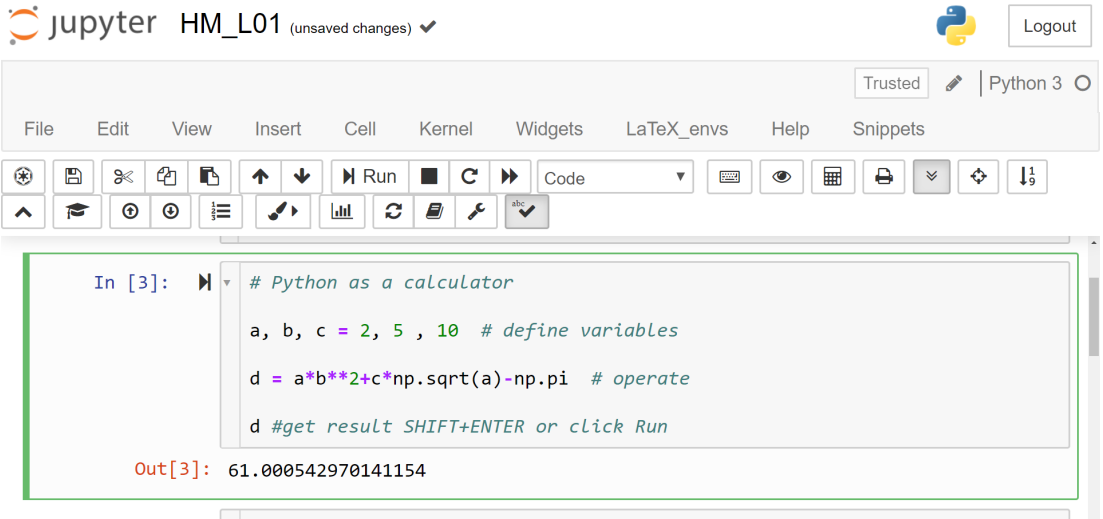
In this block `numpy` and `matplotlib.pyplot` is imported to **JUPYTER**

Also a single function of a package can be imported – in second line the function `erf` is imported

Interacting with JUPYTER notebook

For a **basic** operation (e.g., as a `CALCULATOR`) in **JUPYTER** notebook:

1. Variables (e.g., `a, b, c`) are first defined.
2. Operation using `Operators` and variables are stated.
3. Result is printed when `OUTPUT` variable is put (see last line)
4. Code is **executed** with `Run` or using keybboard.



NOTE: The interactivity between In [3] and Out [3] in the code-block can be repeated only for the particular cell.

Interacting with JUPYTER notebook

For a complex operation (e.g., defining own `FUNCTION`) in **JUPYTER** notebook:

We will learn these by solving our own problems- as we progress with the course.

Here is a basic example how it all works.

Definition parts

jupyter HM_L01 Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets LaTeX_envs Help Snippe

Run Code

In [5]:
```
#Defining your own function

def eq1(x, y, co, v, Dm):
    return co/2*(1-erf(y/(np.sqrt(4*Dm*(x/v)))))
```

In [9]:
```
# using your own function
x = np.linspace(0.1,100,10000)
y = 2
v = 1
Dm = 1
label = ["Co = 1", "Co = 2", "Co = 3", "Co = 4", "Co = 5"]

for co in [1, 2, 3, 4, 5]:
    cm = eq1(x,y,co, v, Dm)
    plt.plot(x, cm)
    plt.legend(label)
    plt.ylabel(r'$C_m$ (mg/L)'), plt.xlabel(r'$x$ (m)')
    plt.grid(), plt.savefig("fig_L1.pdf")

print(cm[5])
```

Execution and Result parts

jupyter HM_L01 Last Checkpoint: an hour ago (autosaved)

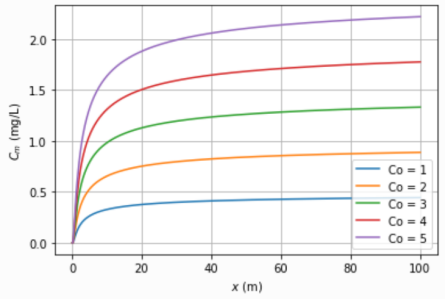
File Edit View Insert Cell Kernel Widgets LaTeX_envs Help

Run Code

```
plt.legend(label)
plt.ylabel(r'$C_m$ (mg/L)'), plt.xlabel(r'$x$ (m)')
plt.grid(), plt.savefig("fig_L1.pdf")

print(cm[5])
```

```
0.0006504346064969968
```



Next steps with Python and JUPYTER

Python is **HUGE**. Becoming a **Python** expert or a programmer is not our **GOAL**. Here we want to make our **HYDRAULICS MODELLING** computation easy. For that a very small part of a much smaller part of **Python** called **SCIENTIFIC PYTHON** is sufficient.

So we focus on following package (mostly **SELF STUDY**)

1. `Numpy`: For matrix and numerical works (see Numpy docs)
2. `Matplotlib`: For plots and visualizations (see Mat docs)
3. `Scipy`: For computations and modelling (see Scipy docs)
4. `Sympy`: For symbolic calculations (see Sympy docs)
5. `Pandas`: For database manipulation (see Pandas docs)
6. `Statsmodels`: For statistical modelling (see docs here)
7. and more as required ...

Python is easy. . .,

Let us learn to work in JUPYTER environment

We will focus on `Numpy`, `Scipy` and `Matplotlib` for our computation and modelling requirements.