Computational Science and Scientific Computing Workshop

Data Programming - Python as a scientific computing tool

Elliot S. Menkah, Ph.D
Daniella N. Apeadu

National Institute for Mathematical Sciences
Kwame Nkrumah University of Science and Technology

October 13, 2025 - October 15, 2025

Python

Why Python?

- It is interpreted and NOT compiled
  - E.g. of Compile languages are C/C++, FORTRAN, etc.
- It's a dynamically-typed language.
- It can be used interactively.
- Syntax is simple, elegant and easily readable.
- Free and open source.
- It's powerful due to its ecosystem of libraries.

Python

Why Python?

- It is interpreted and NOT compiled
  - E.g. of Compile languages are C/C++, FORTRAN, etc.
- It's a dynamically-typed language.
- It can be used interactively.
- Syntax is simple, elegant and easily readable.
- Free and open source.
- It's powerful due to its ecosystem of libraries.

Python

Why Python?

- It is interpreted and NOT compiled
  - E.g. of Compile languages are C/C++, FORTRAN, etc.
- It's a dynamically-typed language.
- It can be used interactively.
- Syntax is simple, elegant and easily readable.
- Free and open source.
- It's powerful due to its ecosystem of libraries.

Python is versatile.

- Download information from a web page.
- Manipulate tests to extract and create information.
- Animate a world in 3D.
- Process huge data sets.
- Make publication-quality graphics.

Which version of Python should I use?

- Currently two(2) versions: 2.7 and 3.7
    - Some packages still work **only** with 2.7
- Versions: 2.7 is deprecated
- Recommend you use version 3.7

Which version of Python should I use?

- Currently two(2) versions: 2.7 and 3.7
  - Some packages still work **only** with 2.7
- Versions: 2.7 is deprecated
- Recommend you use version 3.7

Which version of Python should I use?

- Currently two(2) versions: 2.7 and 3.7
  - Some packages still work **only** with 2.7
- Versions: 2.7 is deprecated
- Recommend you use version 3.7

Which version of Python should I use?

- Currently two(2) versions: 2.7 and 3.7
    - Some packages still work **only** with 2.7
- Versions: 2.7 is deprecated
- Recommend you use version 3.7

ipython Notebook

Why ipython Notebook?

- Gives us a computational notebook with lots of inclusions

- Source code in python and other languages
- Rich text
- Equations written in Latex
- Ready output of results
- Graphics
- Multimedia

ipython Notebook

Why ipython Notebook?

- Gives us a computational notebook with lots of inclusions

- Source code in python and other languages
- Rich text
- Equations written in Latex
- Ready output of results
- Graphics
- Multimedia

ipython Notebook

Why ipython Notebook?

- Gives us a computational notebook with lots of inclusions

- Source code in python and other languages
- Rich text
- Equations written in Latex
- Ready output of results
- Graphics
- Multimedia

ipython Notebook

Why ipython Notebook?

- Gives us a computational notebook with lots of inclusions

- Source code in python and other languages
- Rich text
- Equations written in Latex
- Ready output of results
- Graphics
- Multimedia

ipython Notebook

Why ipython Notebook?

- Gives us a computational notebook with lots of inclusions

- Source code in python and other languages
- Rich text
- Equations written in Latex
- Ready output of results
- Graphics
- Multimedia

ipython Notebook

Why ipython Notebook?

- Gives us a computational notebook with lots of inclusions

- Source code in python and other languages
- Rich text
- Equations written in Latex
- Ready output of results
- Graphics
- Multimedia

ipython Notebook

Why ipython Notebook?

- Gives us a computational notebook with lots of inclusions

- Source code in python and other languages
- Rich text
- Equations written in Latex
- Ready output of results
- Graphics
- Multimedia

Installing Python.

- Alternate: package manager '-apt-get' on Linux or 'brew' on Mac to install python
- Anaconda

How do I run python?

#!/bin/(**bash or zsh**)
$ python
Python 3.6.7 — packaged by conda-forge — (default, Nov 6 2019, 16:03:31)
Type "help", "copyright", "credits" or "license" for more information.
>>>

This is mainly good for running scripts.

#!/bin/bash/zsh
$ ipython
Python 3.6.7 — packaged by conda-forge — (default, Nov 6 2019, 16:03:31)
Type 'copyright', 'credits' or 'license' for more information.
IPython 7.10.2 – An enhanced Interactive Python. Type '?' for help.
In [1]

Anaconda - Conda virtual environment

- **exclusive environment**
- **reinstall anaconda**
- **package dependencies resolution**

Download anaconda via the link: https://www.anaconda.com/distribution/ and download the installer for your respective OS [Linux , mac , windows]

Create an environment:

```
conda create <envname>
Eg.
conda create scim561
```

Connect to environment

```
conda activate scim561
```

Installing packages into an environment

```
conda install <package>
Eg.
conda install matplotlib
```

Deactivate/disconnect from present working environment:

Python Basics

print function, variables, operators

Interpreter - strings and print() function

Print functions and strings:

```
>>> print("Hello World")
Hello World

```

Use double outer quotes (" ") over single outer quotes (' ')

```
>>> print('We\'re here')
We're here

```

to avoid complications.

```
>>> print("We're here")
We're here

```

Interpreter - Variable assignment and Data types

Variables take on the data type of the values being assigned to them.

```
>>> var0 = "hello"
>>> var1 = 7
>>> var2 = 5.2
>>> var3 = True
```

String Variable:

```
>>> print(var0)
hello
>>> type(var0)
<type 'str'>
```

Integer Variable:

```
>>> print(var1)
7
>>> type(var1)
<type 'int'>
```

Interpreter - Variable assignment and Data types

Variables take on the data type of the values being assigned to them.

Floating point Variable:

```
>>> print(var2)
5.2
>>> type(var2)
<type 'float'>

```

Boolean Variable:

```
>>> print(var3)
True
>>> type(var3)
<type 'bool'>

```

Python Operators

Special symbols that carry out arithmetic or logical computation.

Arithmetic Operators

```
+ addition
- substraction
* multiplication
/ division
\% Modulos
// Floor division
** Exponential

```

Logical Operators

```
= assignment operator
== Equal to
< less than
> greater
<= less than or equiv.
>= greater or equiv.
and
or
not

```

Exercises 1

Given an initial velocity, **u**, as 10.2 ms^{-1}, an acceleration, **a**, of 10.01 ms^2 and a time, **t**, of 4 seconds, using the python programming language, write a code to compute the final velocity of a moving particle with the following formulation
$v = u + at$.

data storage, loops, len and range, if statements

Interpreter - List, Tuples and Dictionaries

```
>>> x = ["Hey", "you", 5, 8.7]
>>> y = ("hello", "hi", "you")
>>> w =  {"foo": 1.0, "bar": 2.0 }
>>> print(type(x))
>>> <class 'list'>
>>> print(type(y))
>>> <class 'tuple'>
>>> print(type(w))
>>> <class 'dict'>
```

Empty list:

```
>>> x = []
>>> x
[ ]
```

Interpreter - List, Tuples and Dictionaries

Indexing and memory location:
Memory locations for storing data in list and tuples are indexed so that one could access data stored in a specific memory locations.

NB: By default, index locations begin from zero (0).

```
>>> z = [2, 3, 4, 5]
>>> num0 = z[0]
>>> print(num0)
2

```

Interpreter - List, Tuples and Loops

Loops, List and **range**:

```
>>> for i in z:
...     print(i)
...

```

```
2
3
4
5

```

range & **len** intrinsic functions

```
>>> range(4)
range(0,4)
>>> len(z)
4

```

0 = Starting index
4 = Total no. of numbers
4 = Number of elements in list **z**.

Interpreter - List, Tuples and Loops

range and **len** can be combined and used in loops:

```
>>> for i in range(len(z)):
...     print(i)
...
```

```
0
1
2
3
```

len gives length of list z, that is, 4.
range gives 4 integers used as indexes starting from index 0.

Interpreter - While loops and Boolean

while loops, **if** statements and **boolean**

```
>>> a = True:
>>> print(a)
True
>>> res = 0
>>> while  (a):
...     res += 1
...     print(res)
...     if (res >= 10):
...         a = False


```

```
1
2
3
4
...

```

boolean **a** changes to False and it is used to terminate loop in the condition test section

modules

Modules - import, help, dir

There are lots of libraries in Python that can be imported to use rather than having to build your own. This makes life much easier.
E.g. **math**

```
>>> import math

```

Docs of modules can be viewed with the **help** and **dir** methods.

```
>>> help(< module >)
>>> help(math)
...
...
...
>>> dir(math) or >>> print(dir(math))

```

help gives a comprehensive documentation of the module.
dir gives you the symbols contained in the method concerned.

Modules - import, help, dir

```
>>> help(math.log)
...
...
...
```

import math place the math class in current environment.

```
>>> math.log(10)
2.3025
>>> math.cos(2 * math.pi)
1.0

```

Modules - More on **import**

Partial or selective importation of modules.

In the event of wanting to import only a few symbols into your namespace, the **from** statement is made use of.

```
>>> from math import < symbol or method >
>>> from math import cos
>>> cos(90)
-0.448073616129170l

```

Multiple methods can be imported

```
>>> from math import cos, pi
>>> cos(2 * pi)
1.0

```

Plotting - Matplotlib

Matplotlib

```
import matplotlib.pyplot as plt

plt.plot(X_data, Y_data)
plt.title("Title of plot")
plt.xlabel("X Axis Lable")
plt.ylabel("Y axis Label")

plt.savefig("NameOfFile.png")
```

Listing: Plottin with Matplotlib

functions

Functions

Functions in Python are defined by the keyword **def**

```
>>> def func(x):
...     res = x + 1
...     return res
...
>>> d = func(4)
>>> d
5
```

Python scripts

Script

```
#! /usr/bin/python

print("Hello World")

```

Terminal

```
python3 hello.py
```

Exercises 2

Convert your code from Exercise 1 into a function (that returns a value), where the initial condition, **u**, and the time, **t**, are arguments.

Exercises 3 - Algorithm Development

Exercise A: Multiples of 3 & 5

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Implement an algorithm, with Python, to find the sum of all the multiples of 3 or 5 below 1000.

Exercise B: Fibonacci sequence

Each new term of the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms of the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

arrays and multidimensional vectors

Handling Arrays & Multidimensional Vectors

Vector Operation

$$
\begin{aligned}
\vec{a} \cdot \vec{b} &= \sum_{i=0}^{N} a_i b_i \\
&= \begin{pmatrix} 20 & -3 & 5 \end{pmatrix} \begin{pmatrix} 15 \\ -2.249 \\ 1 \end{pmatrix} \\
&= 20(15) - 3(-2.249) + 5(1) \\
&= 300 + 6.747 + 5 \\
&= 311.747
\end{aligned}
$$

Handling Arrays & Multidimensional Vectors

Multidimensional Arrays

$$\begin{bmatrix} 20 & 15 & 10 & 45 \\ -3 & -2.249 & 7 & 1.751 \\ 5 & 1 & 3 & 9 \end{bmatrix} = \begin{pmatrix} 20 \\ -3 \\ 5 \end{pmatrix} \begin{pmatrix} 15 \\ -2.249 \\ 1 \end{pmatrix} \begin{pmatrix} 10 \\ 7 \\ 3 \end{pmatrix} \begin{pmatrix} 45 \\ 1.751 \\ 9 \end{pmatrix}$$

file I/O, exceptions and assertions

File I/O

keyword: **open**

```
>>> fh = open("demofile.txt", "a")
>>> fh.write("My data file \n")
>>> fh.write("Results: %d", res)
>>> fh.close
```

Exceptions and Assertions

This is a way to handle expected and unexpected errors.

1. Exceptions Handling
2. Assertion

```
try:
# Runs First
< code >
except:
# Runs if exceptions occurs in try block
< code >
else:
# Executes if try block succeeds.
< code >
finally:
# This code always runs executes.
< code >

```

Exceptions and Assertions

Exception Example

```
def read_file(path):
""" Return the content of a file at path"""
try:
fh = open(path, mode="rb")
data = f.read()
return data
except FileNotFoundError as err:
raise
else:
fh.close
finally:
print("Leaving file read routine")

```

Python Basics - End

End of Basics.
Questions ?
Review

Numerical and Scientific Python
Numpy and Scipy libraries

Numerical Python - NumPy

Arrays could be made from:

1. Python list or tuples
2. Using functions that are dedicated to generating numpy arrays, such as *arange*, *linspace*, etc.
3. Reading data from files

```
from numpy import as np
v = array([1,2,3,4])
-----
[1,2,3,4]

```

```
M = np.array([[1, 2], [3, 4]])
-----
array([[1, 2],
[3, 4]])

```

Exercises 4

Using the python programming language, write a code that implements the solution or finds the roots of the non-linear equation:
$3x^2 + 2x + 1 = 0$
using the

1. Bisection Method
2. Newton-Raphson's Method
3. Secant Method

as separate functions.

Classes

Classes in Python are defined by the keyword **class**

```
>>> class myfunctions:
...
...     def add(x):
...         res = x + 2
...         return res
>>>
>>> yy = myfunctions.add(7)
>>> yy
9

```

End of talk

Thank you