Installation to Python 3 Programming: Installation and Basic Coding

Muhammad Izham

Universiti Malaysia Perlis

izham@unimap.edu.my
sugita5019@gmail.com
https://github.com/izham-sugita

Figure 1: Python's logo.

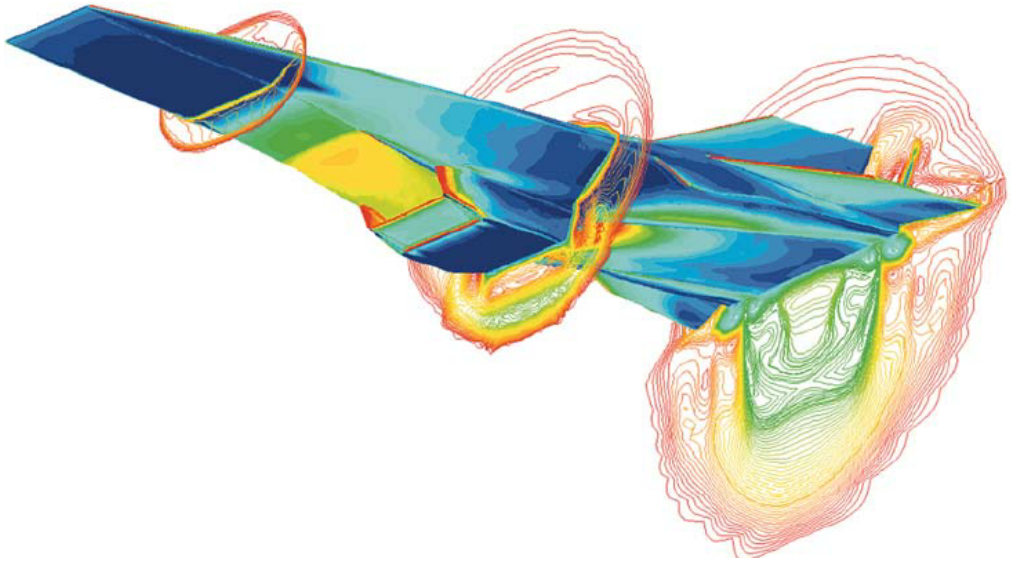


Figure 2: CFD result of HyperX at Mach 7 from NASA

A Little Background

- Created by: Guido van Rossum, 1989-1991
- Why: The creator wanted something easy to use.
- Is it really that easy? Yes (and no)
- Very readable with little memory management.
- `https://people.sc.fsu.edu/~jburkardt/`
- `https://docs.python.org/3/tutorial/index.html`
- `https://www.tutorialsteacher.com/python`
- `https://www.w3schools.com/python/default.asp`
- `https://www.tutorialspoint.com/python/` ⟵ great place to start.

A Little More About Python

- Python2 $\longrightarrow$ Python3; support for Python2 will end 2020.
- This workshop is exclusively on Python3 $\longrightarrow$ refer to just Python
- Python is fully object oriented. Everything is considered object.
- Famous for AI and machine learning $\longrightarrow$ Pytorch, Keras, TensorFlow
- Interpreter language but can be compiled $\longrightarrow$ Cython, Numba
- Very well documented. Every module/libraries are documented online.
- Package management by package installer $\longrightarrow$ pip, pip3
- pip $\longrightarrow$ `https://pypi.org/project/pip/`
- **Py**thon **P**ackage **I**ndex $\longrightarrow$ pypi, `https://pypi.org/`
- `https://github.com/` $\longleftarrow$ another place to look.

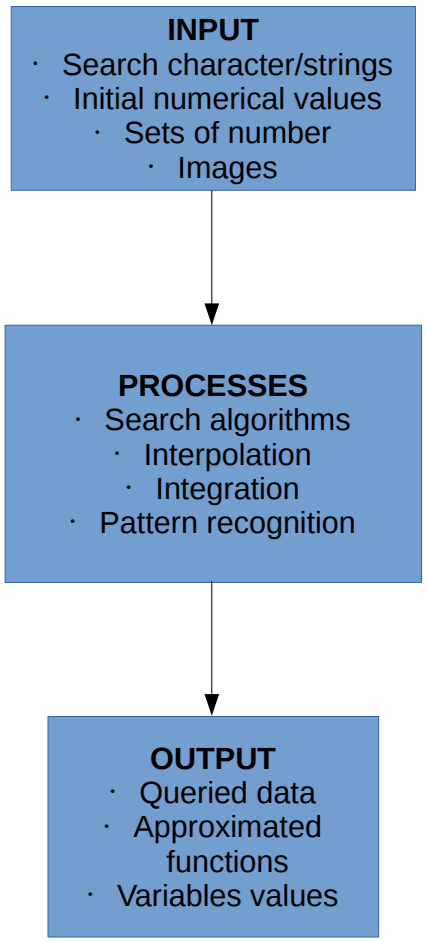
About Anaconda

- No, its not a different programming language.
- Anaconda is a *complete environment* for Python programming.
- Most major scientific package (NumPy, SciPy etc) are included.
- Package installer *conda*

```
user@pc-name:~/conda install any_package
```

- `https://www.anaconda.com/`

About programming



About programming

- Coding paradigm
  - programming language == English (sorry, Mandarin not required)
  - syntax is based on English
  - coding is a reduction of English instructions
- Syntax must be remembered
  - read the manual $\longrightarrow$ documentations are vital
  - memorize THE MOST COMMONLY USED syntax only
  - good algorithm will always beats bad algorithm
- I don't remember every syntax so you have to bare with me

Installation

- Download from `https://www.python.org/`
- For Ubuntu download from repository:

```
user@pc-name:~/apt install python3
```

- For Windows, download from `https://www.python.org/downloads/windows/`
- For Ubuntu installing packages:

```
user@pc-name:~/pip3 install numpy
```

- For Windows installing packages if $C:\Python\Scripts\$ is in the path:

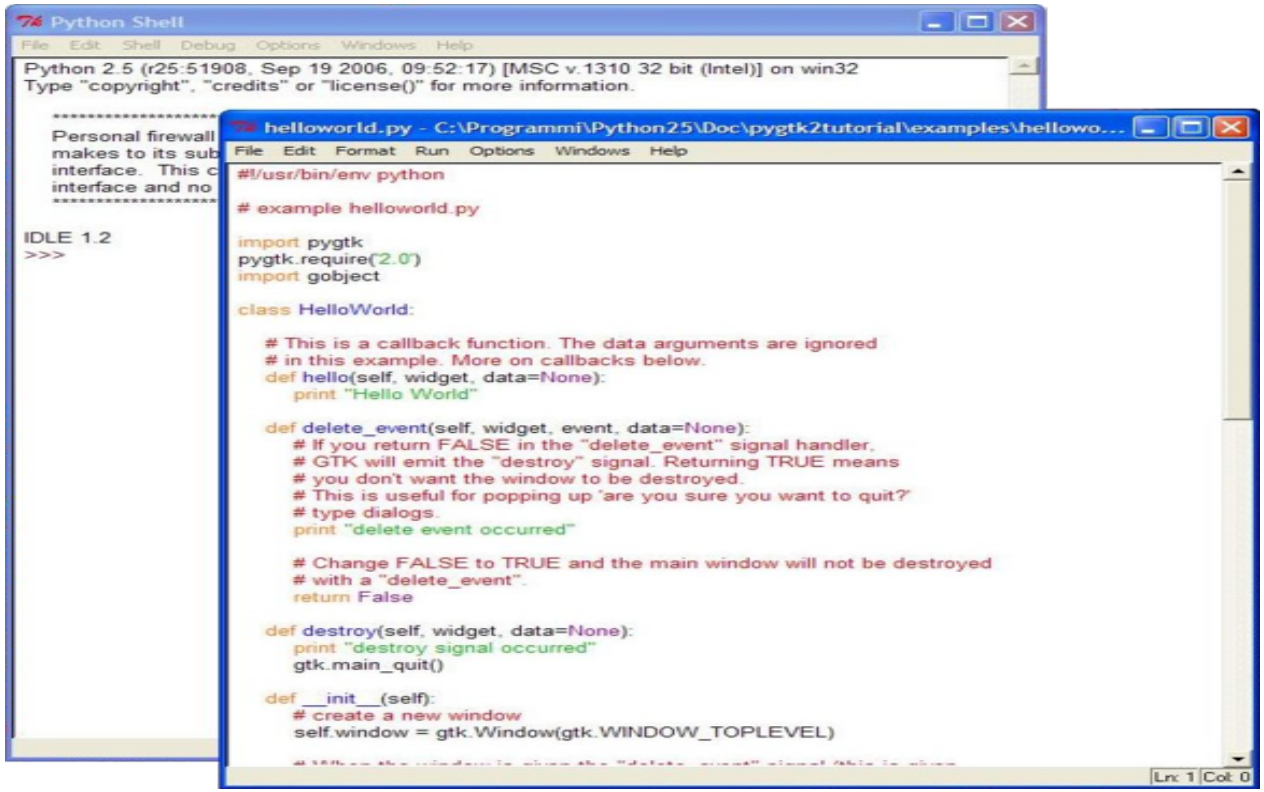
```
C:\User\dummy\> pip3 install numpy
```

- For Windows installing packages:

```
C:\User\dummy\> python3 -m pip3 install numpy
```

Introduction

- For Windows, the default editer is IDLE



Simple Snippets

Hello, Python!

- Symbols to remember.

```
#everything about the basics

'''
This is how you put a long comment.
: <- colon
; <- semi-colon
() <-parentheses
[] <-brackets
{} <- braces
# <- hash sign
' <- apostrophe
" <- quote mark
.....
Thats about everything you need to know/remember
'''
```

file: py-hello.py

Hello, Python!

- The print() function.

```
1  print("Hello,world!")
2  print("Hello, Python 3\n") # "\n"; next line
3  money=4.0
4  print("I have %f B, in a yacht somewhere"%money)
5  var0 = 1
6  var1 = 1.07
7  print("To print float number %f and integer %d"%(
       var1, var0))
8  str1 = "Another way "
9  str2 = "to add something."
10 print(str1+str2)
11 print(str1+"\t"+str2) # "\t" for tab spacing
```

file: py-hello.py

Basic I/O including files.

- Basic I/O from terminal.

```
1  #Basic terminal input
2  var0 = input("Enter something:")
3  print(var0) #input by default is STRING ONLY!
4
5  var1 = input("Enter integer: ")
6  var1 = int(var1)
7  print(var1, var1+var1)
8  print(var1, var1, var1, var1) #you can print a lot
9
10 var2 = input("Enter real number: ")
11 var2 = float(var2)
```

file: py-basic-io.py

Basic I/O including files.

- Write to file and close.

```
fo = open("test.txt","w+")
fo.write("Test file\n")
fo.close()
```

- Write to file with different access mode.

```
fo = open("test.txt","a+")
anim_list = ["cat","tiger","bear","elephant","mouse
    ","dear"]

for animal in anim_list:
    print(animal)
    fo.write(animal+"\n")

fo.close()
```

file: py-file-io.py

Loop in Python

- for-loop

```
1 #Example of simple loop
2 import numpy as np
3 imax = 64
4 a = np.ndarray(shape=(imax), dtype=float)
5 for i in range(imax):
6         a[i] = 1.0
```

Loop in Python

- Loop with string data.

```
1 words = ['I', 'me', 'you', 'him']
2 for w in words:
3   print(w, len(w))
```

Loop in Python

- while-loop.

```
1  i=1
2  while i<6:
3      print(i)
4      i +=1
5
6  i=1
7  while i<6:
8      print(i)
9      if i==4:
10         break
11     i +=1
```

file: py-while.py

More precise loop in Python

- for-loop for forward-backward sweep.

```
1 #print precise
2 for i in range(6,1,-1):
3     print("a[%d]=%.4f"%(i,a[i]))
4
5 print()
6 for i in range(1,6):
7     print("a[%d]=%.4f"%(i,a[i]))
```

file: py-precise-loop.py

Applications: Important Tools and Sample Codes

NumPy module

- Specific module for numerical simulations.
- Very excellent for array manipulation.
- Documentation from `https://www.numpy.org/`

SciPy module

- Scientific calculations module.
- Lots of functions for scientific computation.
- Documentation from `https://www.scipy.org/`
- Specific manuals
  `https://docs.scipy.org/doc/scipy-1.3.0/reference/`

Matplotlib module

- Very efficient module for plotting almost everything.
- Capable for plotting graphs and images.
- Documentation from
  `https://matplotlib.org/3.1.0/index.html`

```
1 import matplotlib.pyplot as plt
2 import matplotlib.image as mpimg
3 import numpy as np
4
5 img = mpimg.imread('Lenna-test.png')
6 imgplot = plt.imshow(img)
7 plt.show()
```

file: py-matplotlib.py

pandas module

- Python data analysis module.
- The go-to module for data analysis.
- Documentation from `https://pandas.pydata.org/`

Numba module

- Python module for high performance computing.
- Just-In-Time (JIT) compiler for faster computation.
- Documentation from `https://numba.pydata.org/`

*Numba with CUDA

- GPU computing implementation using Numba-CUDA.
- Very common in AI and DeepLearning implementation.
- Example code:

```
1 import numpy as np
2 import numba.cuda
3 numba.cuda.api.detect()
4 numba.cuda.cudadrv.libs.test()
```

file: py-numba-cuda.py

*Special settings are required.

Numba with CUDA

- Example output for device detection.

```
sugita@alienware:~/codes/python3-test/python3-tutorial$ python3 py-numba-cuda.py
Found 3 CUDA devices
id 0     b'GeForce GTX 1080'                              [SUPPORTED]
                    compute capability: 6.1
                         pci device id: 0
                            pci bus id: 2
id 1     b'GeForce GTX 1080'                              [SUPPORTED]
                    compute capability: 6.1
                         pci device id: 0
                            pci bus id: 3
id 2          b'Quadro P400'                              [SUPPORTED]
                    compute capability: 6.1
                         pci device id: 0
                            pci bus id: 1
Summary:
       3/3 devices are supported
Finding cublas from CUDA_HOME
       located at /usr/local/cuda-9.2/lib64/libcublas.so.9.2.174
       trying to open library...       ok
Finding cusparse from CUDA_HOME
       located at /usr/local/cuda-9.2/lib64/libcusparse.so.9.2.148
       trying to open library...       ok
Finding cufft from CUDA_HOME
       located at /usr/local/cuda-9.2/lib64/libcufft.so.9.2.148
       trying to open library...       ok
Finding curand from CUDA_HOME
       located at /usr/local/cuda-9.2/lib64/libcurand.so.9.2.148
       trying to open library...       ok
Finding nvvm from CUDA_HOME
       located at /usr/local/cuda-9.2/nvvm/lib64/libnvvm.so.3.2.0
       trying to open library...       ok
Finding libdevice from CUDA_HOME
       searching for compute_20...     ok
       searching for compute_30...     ok
       searching for compute_35...     ok
       searching for compute_50...     ok
sugita@alienware:~/codes/python3-test/python3-tutorial$
```

Figure 3: Numba's device detection.

file: py-numba-cuda.py

Numba with CUDA

- Example for CUDA kernel using numba

```
1  # CUDA kernel
2  @cuda.jit
3  def matmul(A, B, C):
4      """Perform matrix multiplication of C = A * B
5      """
6      row, col = cuda.grid(2)
7      if row < C.shape[0] and col < C.shape[1]:
8          tmp = 0.
9          for k in range(A.shape[1]):
10             tmp += A[row, k] * B[k, col]
11         C[row, col] = tmp
```

file: py-numba-cuda-mxm.py

Finite difference method for heat equation

- The 1D equation : $\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2}$

```
1  import numpy as np
2  from numba import jit, njit
3  import time
4
5  @jit(nopython=True, nogil=True, cache=True, parallel=True)
6  def theloop(itermax,imax,icenter,coeff,dt,fx,fxnew,peak,tminus):
7      for iter in range(itermax):
8          for i in range(1,imax):
9              fxnew[i] = fx[i] + coeff*( fx[i-1] - 2.0*fx[i] + fx[i+1] )
10         peak.append(fxnew[icenter])
11         tminus.append((iter+1)*dt)
12         fx = fxnew
```

file: 1D-heat-numba.py

Finite difference method for heat equation

- Visualization using matplotlib.

```
1  import matplotlib
2  import matplotlib.pyplot as plt
3  fig, ax = plt.subplots()
4  ax.plot(x, initfx)
5  ax.plot(x,fxnew)
6  ax.set(xlabel='x', ylabel='f(x)',
7         title='Initial value')
8  ax.grid()
9  fig.savefig("test-numba.png")
10 #plt.show()
11
12 #next plot
13 fig0, decay = plt.subplots()
14 decay.plot(tminus, peak)
15 decay.set(xlabel='time', ylabel='f(x)_max',
16         title='Peak decay')
17 decay.grid()
18 fig0.savefig("decay-numba.png")
```

file: 1D-heat-numba.py

Monte-Carlo Simulation

- Monte-Carlo simulation to calculate the value of π
- The code below shows a straightforward implementation.

```
1  ts = time.time()
2  count = 0
3  for i in range(particle):
4      xp = np.random.random()
5      yp = np.random.random()
6      rad = np.sqrt(xp*xp + yp*yp)
7      if rad < 1.0:
8          count +=1
9  te = time.time()
10 normal_time = te-ts
```

file: py-MonteCarlo.py

Monte-Carlo Simulation

- With Numba jit

```
1  from numba import jit
2
3  @jit(nopython=True, cache=True, nogil=True)
4  def mc(particle):
5      count = 0.0
6      for i in range(particle):
7          xp = np.random.random()
8          yp = np.random.random()
9          rad = np.sqrt(xp*xp + yp*yp)
10         if rad < 1.0:
11             count +=1.0
12     return count
```

file: py-MonteCarlo.py

Monte-Carlo Simulation

- Computation time Numba vs Normal script

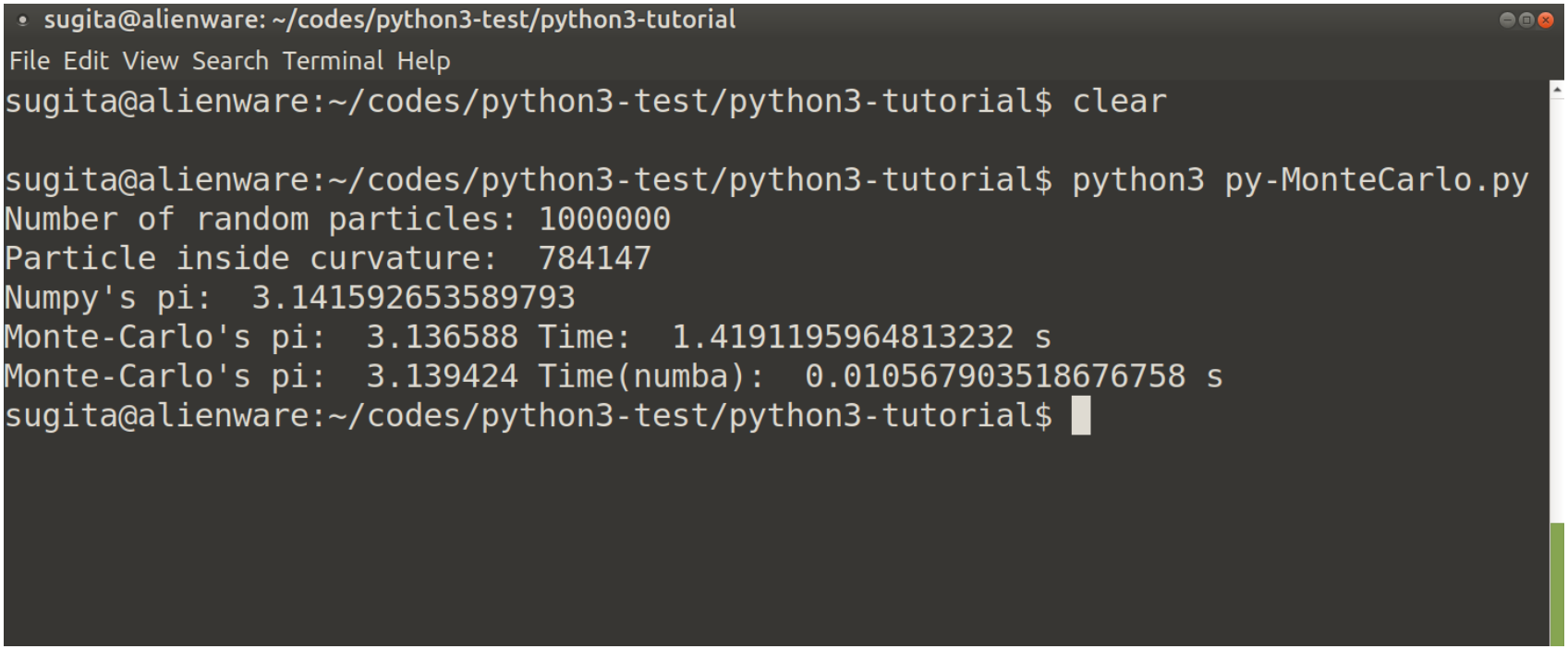


Figure 4: Numba vs normal script computation time.

Thank You!
Questions?