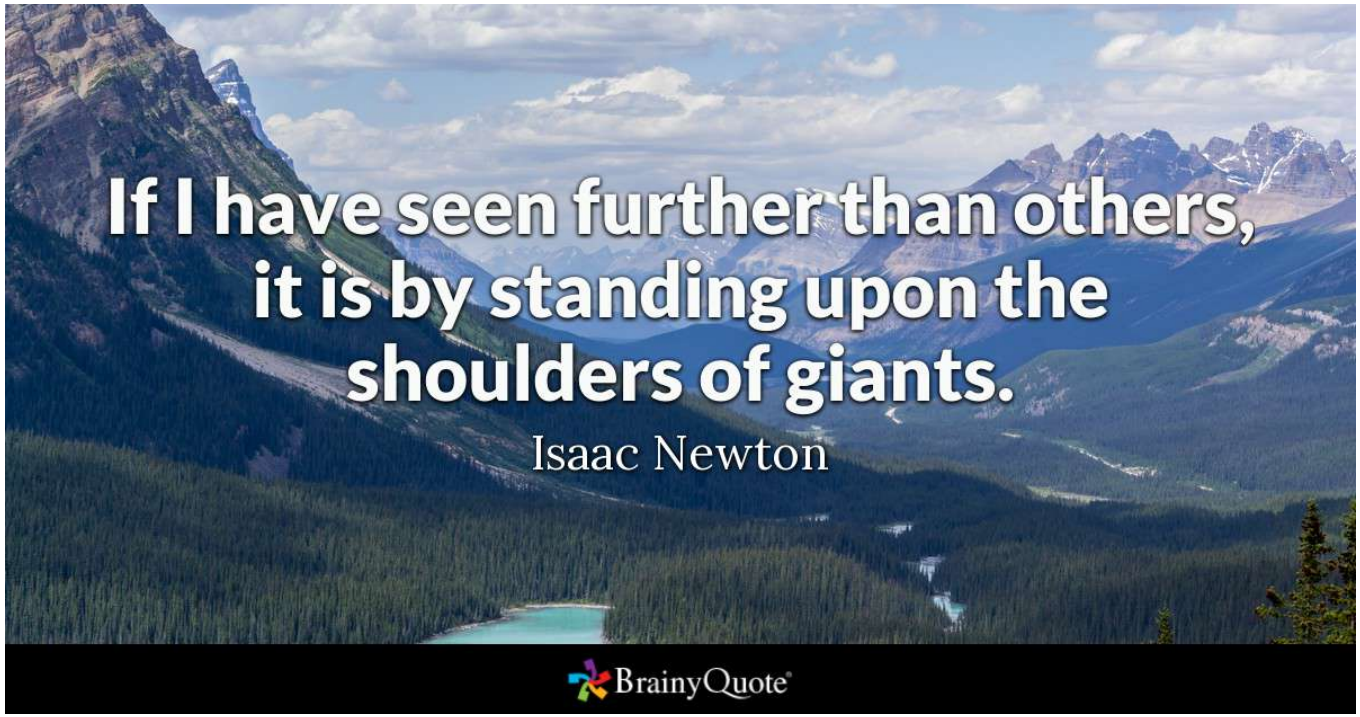
Python for Beginners/Non-Programmers

Muhammad Izham

Universiti Malaysia Perlis

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

Why Python is here



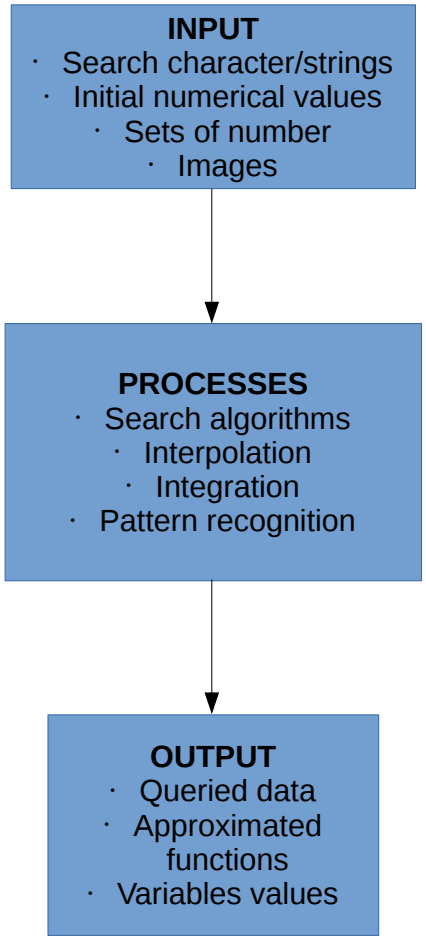
Nope! Not this one!

Overview

Before we get started, does anyone want to get out?

- Simple paradigm of programming
  - what engineers and scientists care about
  - a lawyer and a cop
  - I will get some of the explanations imprecise
- Programming basic concepts
  - What is it REALLY all about?
    - data processing/manipulations
    - programming is a process/steps of processing/manipulating data
  - What is the most basic types of data?
    - characters/strings
    - number/numerics $\longrightarrow$ float/integer

Before we get started, does anyone want to get out?



Before we get started, does anyone want to get out?

- 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 $C:\backslash Python\backslash Scripts\backslash$ is in the path:

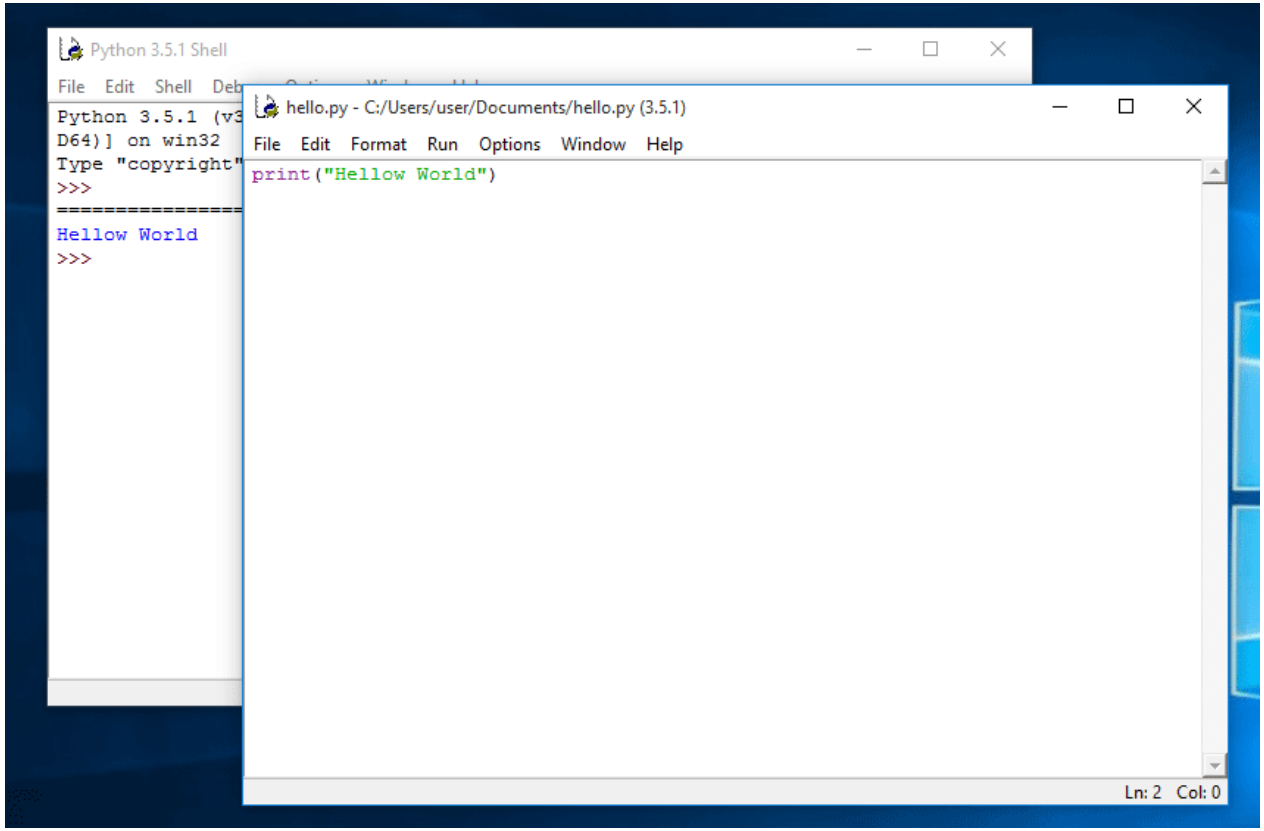
```
C:\User\dummy\> pip3 install numpy
```

- For Windows installing packages:

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

Installation

- For Windows, the default editor is IDLE



For Your Information

- 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.
- 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.

A Little Bit 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/`

The Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one – and preferably only one – obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea – let's do more of those!

The "Hello, World!"

Greetings!!

```
1 print("Hello,Python!")
```

file: helloPython.py

Default/Built-in variables type

- Python has 5 default/built-in variables:
  - character/strings
  - numbers/numeric
  - list/tuple/dictionary $\longrightarrow$ actually a derivative
- The list/tuple/dictionary is a *collection* of character/strings/numbers.
- Since Python treats *everything* as an object, the derivatives data type can be manipulated like the basic data type too.

Variables size

In Python:

```
1  import numpy as np
2  import sys
3  a = 32
4  b = np.int32(a)
5  c = np.int64(a)
6  d = np.float32(a)
7  e = np.float64(a)
8
9  print(sys.getsizeof(b),"bytes")
10 print(sys.getsizeof(c),"bytes")
11 print(sys.getsizeof(d),"bytes")
12 print(sys.getsizeof(e),"bytes")
```

file: py-number.py

Variables size

In C/C++/Fortran:

```c
#include<stdlib.h>
#include<stdio.h>

int main()
{
   int a=32;
   long int b = 32;
   float c = 32;
   double d = 32;
   printf("size of int is %ld bytes\n", sizeof(a));
   printf("size of long int is %ld bytes\n", sizeof(b));
   printf("size of float is %ld bytes\n", sizeof(c));
   printf("size of double is %ld bytes\n", sizeof(d));
}
```

file: datasize.c

Variables size

- In Python, int=28, long int=32, float=32, double=32 bytes.
- Generally a lot more than standard programming language; why?
- In C/C++/Fortran int=4, long int=8, float=4, double=8 bytes
- Curious?

Variables in Python3

Importing a module:

```
1 import numpy as np
```

Print to screen:

```
1 a = 5
2 print("a=%f"%a)
3 a = "Any character"
4 print(a)
```

Taking an input:

```
1 x=int(input("Enter an integer: "))
2 print(x)
```

file: variables.py

Variables in Python3

How to use module:

```
1 x = np.float32(input("Enter a single precision number ")
  )
2 print(x)
3
4 print("\nWill print numbers with different precision\n")
  )
5
6 x = np.float64(np.random.random())
7 print(x)
8 x = np.float32(np.random.random())
9 print(x)
```

file: variables.py

Repetitive task

- 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
```

Repetitive task

- Loop with string data.

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

Control: Asking to do things

- Simple control.

```
1 x = int(input("Enter an integer: "))
2 if x > 0:
3         print("x = %d\n"%(x))
```

file: py-control.py

Control: Asking to do things

- Nested if-elif

```
print("if Statement\n")
x = int(input("Please enter an integer: "))
if x < 0:
  x = 0
  print("Negative value will turn to ZERO")
elif x == 0:
  print("Zero")
elif x == 1:
  print("Single")
```

file: py-control.py

Container: List, Dictionaries, Set, Tuples

Container: List

- A list can contains any type of variable
- Unlike the normal practice of array where an array contains just one type of variable

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])  # Prints "[3, 1, 2] 2"
print(xs[-1])     # Negative indices count from the end
    of the list; prints "2"
xs[2] = 'foo'     # Lists can contain elements of
   different types
print(xs)         # Prints "[3, 1, 'foo']"
```

file: py-container.py

Container: List, Dictionaries, Set, Tuples

Container: Dictionaries

- Key paired with value(s)
- Key is unique; value(s) may be changed.

```
dic = {'Name':'Sugita', 'Age': 40, 'Occupation':'
    Lecturer'}
print('Name: ', dic['Name'])
print('Age: ', dic['Age'])
print('Occupation: ', dic['Occupation'])
```

- Updating dictionary

```
#update dictionary
dic['Qualification'] = 'PhD'
print('Qualification: ', dic['Qualification'])
```

file: py-container.py

Container: List, Dictionaries, Set, Tuples

Container: Set

- A set is a collection which is **unordered and unindexed**.
- A set cannot be access in order or changed in value, but you can add to it.

```
#a set
myset = {"apple", "banana", "cherry"}
print(myset)
```

- Add to set using *add()* or *update()* method.

```
#add to set
myset.add("jackfruit")

#update to add more than one
myset.update(["papaya","mango","durian"])
```

file: py-container.py

Container: List, Dictionaries, Set, Tuples

Container: Tuples

- A tuple is a collection of data that are *ordered and* **unchangeable**
- Very useful for protecting a set of parameters from careless over-write.

```
#tuple
mytuple = ('apple','papaya','banana')
print(mytuple[0])
```

file: py-container.py

Class in Python3

- Variables in class are public by default.

Defining a class:

```
#define a class
class vehicle:
    name = ""
    kind = "car"
    color = ""
    value = 100.0
    def description(self):
        desc_str = \
        "My %s is a %s %s worth $%.2f." \
        %(self.name, self.color, self.kind, self.value)
```

file: py-class0.py

Class in Python3

- Variables can be made private.

Defining a class:

```
1 class employee:
2     #functions to initiate
3     def __init__(self, name, salary):
4         self.__name=name      #protected attribute,
      hence private
5         self.__salary=salary #protected attribute
```

file: py-class0.py

Class in Python3

- Initiating class.

```
1  class Person:
2      def __init__(self, name, age):
3          self.name = name
4          self.age = age
5
6      def myfunc(self):
7          print("Hello, My name is "+self.name)
8
9  #Create an object of Person
10 p1 = Person("John",40) #must initiate name and age
```

file: py-class1.py

Function: Modularizing your codes

- Function is very useful to compartmentalize your code(s).
- Function split task $\longrightarrow$ allowing modularity.

```
def functionname(parameters):
  "function comment"
   function_suite
   return [expression]
'''
```

file: py-function.py

Function: Modularizing your codes

- A simple example.

```
1 def printme( str ):
2   "This function prints the string input"
3    print( str )
4    return
```

file: py-function.py

Function: Modularizing your codes

- Function at its very basic $\longrightarrow$ 1 input 1 output

```
1 def foo(arg1):
2         return 2*arg1
```

file: py-function.py

Function: Modularizing your codes

- Creating a function that change the value of its input:

```
1 def sum2( arg1 , arg2 , arg3 ):
2         arg3[0] = arg1[0] + arg2[0]
3         return
```

- Function parameters in Python are pass-by-reference.
- To change the value $\longrightarrow$ mutable object as input parameter e.g. list, numpy's array

file: py-function.py

Function: Modularizing your codes

- Creating a function that change the value of its input:

```
1 def sum4(arg1, arg2, arg3):
2         arg3[:] = arg1[:] + arg2[:]
3         return
```

- Input parameters as numpy's array.

```
1         a = np.ndarray(shape=(1), dtype=int)
2         b = np.ndarray(shape=(1), dtype=int)
3         c = np.ndarray(shape=(1), dtype=int)
4         a[0] = 1
5         b[0] = 2
6         c[0] = 0
7         sum4(a,b,c)
8         print(c[0]) #will return 3 because np.
    ndarray is mutable.
```

file: py-function.py

Conclusion for coding

- Python syntax is easy (relatively) to learn.
- However, mastery WILL require a lot of practice.
- Python paradigm is very suitable for non-developer programmer.

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

Numba with CUDA

- Example for CUDA kernel using numba

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

file: py-numba-cuda-mxm.py

Thank You!
Questions?