Python:

- Open source general-purpose
- Object-Oriented, Procedural, Functional
- Easy to interface with C/ObjC/Java/Fortran/C++
- Great interactive environment

Setup Environment:

- Download and install anaconda:
  https://www.anaconda.com/distribution/
- Python promt
  - **>>>**
- Create a python environment:
  - **>>>** conda create -n nameOfEnvironment python=pythonversion
    eg.
    >>> conda create -n EICT python=3.6
- Activate a python environment
  - **>>>** conda activate nameOfEnvironment
    eg.
    >>> conda activate EICT
- Deactivate a python environment
  - **>>>** conda deactivate
- Install a python package
  - **>>>** conda install packageName
    eg.
    >>> conda install spyder
- Running a python script
  - **>>>** python filename.py

Note: To make a python (*.py) file executable add **#!/usr/bin/env python** to the top of file.

We can edit python code in any text editor and then run it by command “python filename.py” but it will be more convenient if we use an IDE instead. Spyder and pyCharm are some good examples of python IDEs.

Python prompt

Python gives a prompt.

>>>

>>>7+8

15

you can initialize a variable in it:

>>> x=7+8+6

>>>x

21

Also used to initialize a string:

>>> x='tali'

>>>x

'tali'

>>>x=x+x

in case of string it works as a concatenation operator

>>>x

'talitali'

Strings

Can use **“ ”** or **‘ ’** to specify.

“TOFIK ALI” ‘TOFIK ALI’ (Same thing.)

Unmatched can occur within the string.

“Tali’s”

Use triple double-quotes for multi-line strings or strings that contain both **‘** and **“** inside of them:

“““a‘b“c”””

Python Indentation

In java /c / c++, the grouping of the statements is done by using braces. However, in python, this is done by indentation. It is used as a way of grouping the statement.

E.g.

In C

```
if (i%2==0)
{
print("the number is even");
print("thank you");
}
else
{
print("the number is odd");
print("thank you");
}
```

However in python

```
if (i%2==0):
        print("the number is even")
        print("thank you")
else:
        print("the number is odd")
        print("thank you")
```

No semicolons (**;**) for the termination in case of python.

Use a newline (Enter key of a keyboard) to end a line of code.

Use \ when must go to next line prematurely.

No braces { } to mark blocks of code in Python...

Use consistent indentation instead.

• The first line with less indentation is outside of the block.

• The first line with more indentation starts a nested block

• Often a colon appears at the start of a new block.

(E.g. for function and class definitions.)

Types and declaration

Python is flexible in the context of type declaration of a variable before using (it is a necessary part in java / c programming).

The first assignment to a variable creates it.

Variable types don't need to be declared.

Python figures out the variable types on its own.

To get the type of a variable there is a command in python (type)

```
>>> type("TOFIK ALI")
<type 'str'>
>>> type(786)
<type 'int'>
>>>x=786
>>>print(x)
786
```

Comments

• Start comments with # – the rest of line is ignored.

• Can include a "documentation string" as the first line of any new function or class that you define.

```
def myFunction(x, y):
"""This is the docstring. This
function does blah blah blah."""
# The code would go here...
```

Assignment

• Binding a variable in Python means setting a name to hold a reference to some object.
• Assignment creates references, not copies
• Names in Python do not have an intrinsic type. Objects have types.
• Python determines the type of the reference automatically based on the data object assigned to it.
• You create a name the first time it appears on the left side of an assignment expression:

```
>>> x = 3
```

• A reference is deleted via garbage collection after any names bound to it have passed out of scope.

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

Naming Rules

• Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.

```
bob
Bob
_bob
_2_bob_
bob_2
BoB
```

• There are some reserved words:

and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while

Reference Semantics

• Assignment operator (**=**) is used to manipulates references

```
>>>a = b  #does not make a copy of the b, it makes a reference the object b references
```

Warning: Very useful, but beware!

```
>>> a = [1, 2, 3]
 # a references the list [1, 2, 3]
>>> b = a
 # now b references what a references
>>> a.append(4)
```

```
 # it append item 4 in the list that a references
>>> print b
 [1, 2, 3, 4]
 # SURPRISE! It has changed...
```

In Python, the datatypes integer, float, and string (and tuple) are "**immutable**" and lists, dictionaries, user-defined objects are "**mutable**."

immutable	mutable

>>> x = 3
>>> y = x
>>> y = 4
>>> print x
3
	>>>x = some mutable object
	>>>y = x
	>>>make a change to y
	>>>look at x
	x will be changed as well

Operator and operand

The operators +, -, *, / and ** perform addition, subtraction, multiplication, division and Exponentiation.

Order of operation:
P : Paranthesis
E : Exponent
D : Division
M : Multiplication
A : Addition
S : Substract

Operators with the same precedence are evaluated from left to right (except exponentiation).

Modulus operator **%**

```
if (i%2==0):
        print("the number is even")
        print("thank you")
else:
        print("the number is odd")
        print("thank you")
```

Relational operator:

```
x != y    # x is not equal to  y
x > y     # x is greater than y
x < y     # x is less than y
x >= y    # x is greater than or equal to y
x <= y    # x is less than or equal to y
```

Logical operator:

and, or , not

eg.

a%4 == 0 or a%5 == 0

Condition

```
if (i%2==0):
        print("the number is even")
        print("thank you")
else:
        print("the number is odd")
        print("thank you")
```

Chained execution :

```
if (i%2==0):
        print("the number is even")
        print("thank you")
elif i%3==0:
        print("the number is divisible by 3")
        print("thank you")
elif i%5==0:
        print("the number is divisible by 5")
        print("thank you")
```

Nested condition :

```
if i>j:
        print(“i is greater than j”)
        print(“thank you“)
else:
        If i==j:
                print(“i and j are equal”)
                print(“thank you”)
        else:
                print(“i is less than j”)
                print(“thank you”)
```

Loops

While loop

```
while condition:
        Block of code
```

Eg.

```
a=0
while a < 10:
        print(a)
        a += 1
```

Break statement: it is same as in other languages

```
while True:
        n = int( input('enter any value = ') )
        if n>10:
                print(‘number entered is greater than 10’)
                break
```

For loop

```
for item in list:
        Block of code
```

eg.

```
for i in range(5):
        print(i)
```

Continue statement: Skip the execution of loop untill certain condition.

```
counter = 0
while 1:
        if counter < 5:
                continue;
        if counter > 10:
                break;
        counter += 1; # counter+=1 wrong there is a space before and after +=
```

Functions

It is not a good idea to write everything in python prompt. If you are writing a complex set of codes that require a lot of correction over the development period then storing it as a runnable script is more convenient. Inside a script, we can define some of the code segments as a function and then use it as we want.

Syntax of a function prototype as follow

```
def funtionName(argument1, argument2,..., argx=defaultvalue1, argy=defaultvalue2):
        code segment
        return out1, out2,out3
```

Parameter and arguments: Inside the function, the arguments are assigned to
Please go through the assignment 1 to 3.

Importing a function from other python file or package:

```
Import functionNamex from packageNameX
Import functionNamey from fileNameY
Import packageNameX
Import fileNameY

a=functionNamex(arg1,arg2)
b=packageNameX.functionNamex(arg1,arg2)

a=functionNamey(arg1,arg2)
a=fileNameY.functionNamey(arg1,arg2)
```

Matrices with NumPy

- NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array. Using NumPy, a developer can perform the following operations −
    - Mathematical and logical operations on arrays.
    - Fourier transforms and routines for shape manipulation.
    - Operations related to linear algebra. NumPy has in-built functions for linear algebra and

- random number generation.
  NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library). This combination is widely used as a replacement for MatLab, a popular platform for technical computing. However, Python alternative to MatLab is now seen as a more modern and complete programming language. It is open-source, which is an added advantage of NumPy.

- Create numpy array :
  The basic ndarray is created using an array function in NumPy as follows −
  >>>import numpy as np
  >>>a = np.array([1,2,3])
  >>>print(a)

  # more than one dimensions
  >>>import numpy as np
  >>>a = np.array([[1, 2], [3, 4]])
  >>>print(a)

  Each element in ndarray is an object of data-type object specified by dtype argument.
  # dtype parameter
  >>>import numpy as np
  >>>a = np.array([1, 2, 3], dtype = complex)
  >>>print(a)

  The ndarray object consists of a contiguous one-dimensional segment of computer memory, combined with an indexing scheme that maps each item to a location in the memory block.

- Array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.
  # to get the shape of the array
  >>>import numpy as np

```
>>>a = np.array([[1,2,3],[4,5,6]])
>>>print(a.shape)

# this resizes the ndarray
>>>import numpy as np
>>>a = np.array([[1,2,3],[4,5,6]])
>>>a.shape = (3,2)
>>>print(a)

# also you can use reshape
>>>import numpy as np
>>>a = np.array([[1,2,3],[4,5,6]])
>>>b = a.reshape(3,2)
>>>print(b)

# an array of evenly spaced numbers
>>>import numpy as np
>>>a = np.arange(24)
>>>print(a)

To creates an uninitialized array of specified shape and dtype.
>>>import numpy as np
>>>x = np.empty([3,2], dtype = int)
>>>print(x)

To create an array of specified size, filled with zeros.
# array of five zeros. Default dtype is float
>>>import numpy as np
>>>x = np.zeros(5)
>>>print(x)
>>>y = np.zeros((5,), dtype = np.int)
>>>print(y)
```

- To create an array of specified size, filled with ones.
  ```
  # array of five ones. Default dtype is float
  >>>import numpy as np
  >>>x = np.ones(5)
  >>>print(x)
  ```

- asarray(data) :
  This function is similar to numpy.array except for the fact that it has fewer parameters. This routine is useful for converting Python sequence into ndarray.

```
# convert list to ndarray
>>>import numpy as np
>>>x = [1,2,3]
>>>a = np.asarray(x)
>>>print(a)

# dtype is set
>>>import numpy as np
>>>x = [1,2,3]
>>>a = np.asarray(x, dtype = float)
>>>print(a)

# ndarray from tuple
>>>import numpy as np
>>>x = (1,2,3)
>>>a = np.asarray(x)
>>>print(a)

# ndarray from list of tuples
>>>import numpy as np
>>>x = [(1,2,3),(4,5)]
>>>a = np.asarray(x)
>>>print(a)
```

- Indexing and slicing:
  Items in ndarray object follow the zero-based index. A Python slice object is constructed by giving a start, stop, and step parameters to the built-in slice function.

```
>>>import numpy as np
>>>a = np.arange(10)
>>>s = slice(2,7,2)
>>>print(a[s])
```

  Output:
  [2 4 6]

- Then a slice object is defined with start, stop, and step values 2, 7, and 2 respectively. When this slice object is passed to the ndarray, a part of it starting with index 2 up to 7 with a step of 2 is sliced.

  The same result can also be obtained by giving the slicing parameters separated by a **colon operator :** (start:stop:step) directly to the ndarray object.

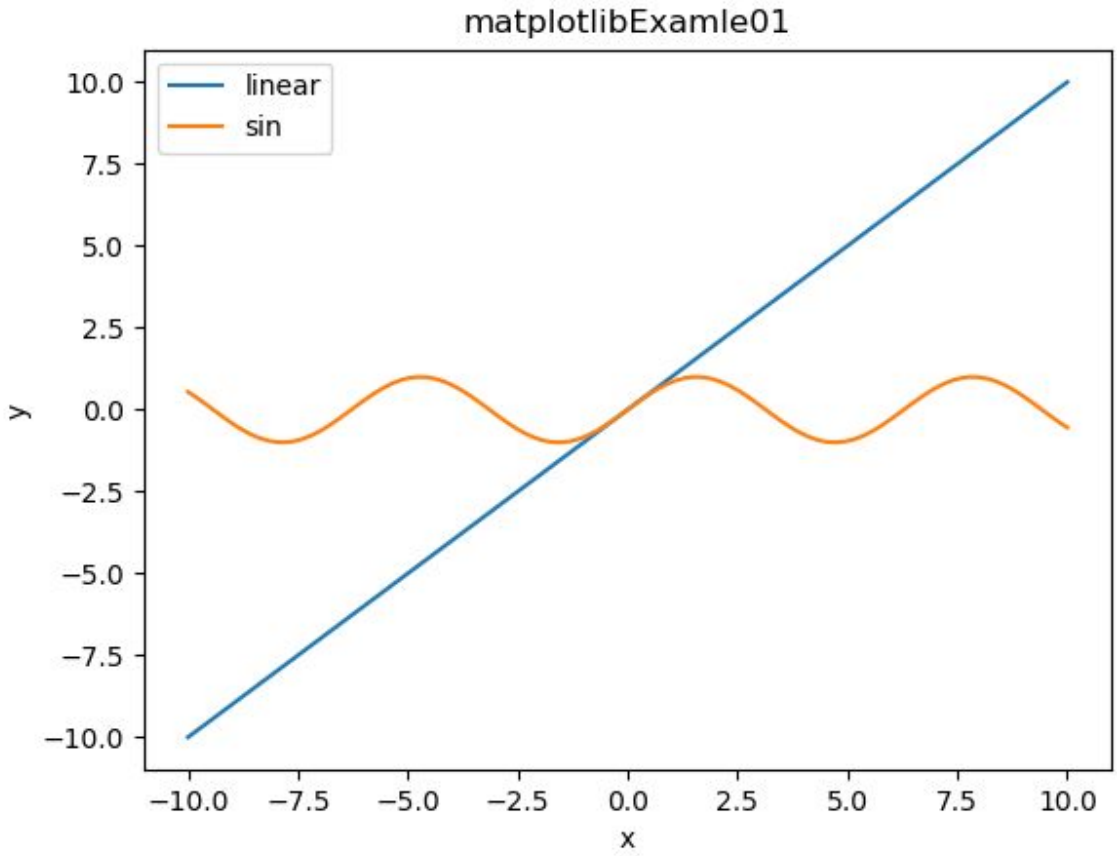
```
>>>import numpy as np
>>>a = np.arange(10)
```

```
>>>b = a[2:7:2]
>>>print(b)
Output:
[2 4 6]

# slice single item
>>>import numpy as np
>>>a = np.arange(10)
>>>b = a[5]
>>>print(b)
output :
5

# slice items starting from index
>>>import numpy as np
>>>a = np.arange(10)
>>>print(a[2:])
Output:
[2 3 4 5 6 7 8 9]

# slice items between indexes
>>>import numpy as np
>>>a = np.arange(10)
>>>print(a[2:5])
Output:
[2 3 4]
```

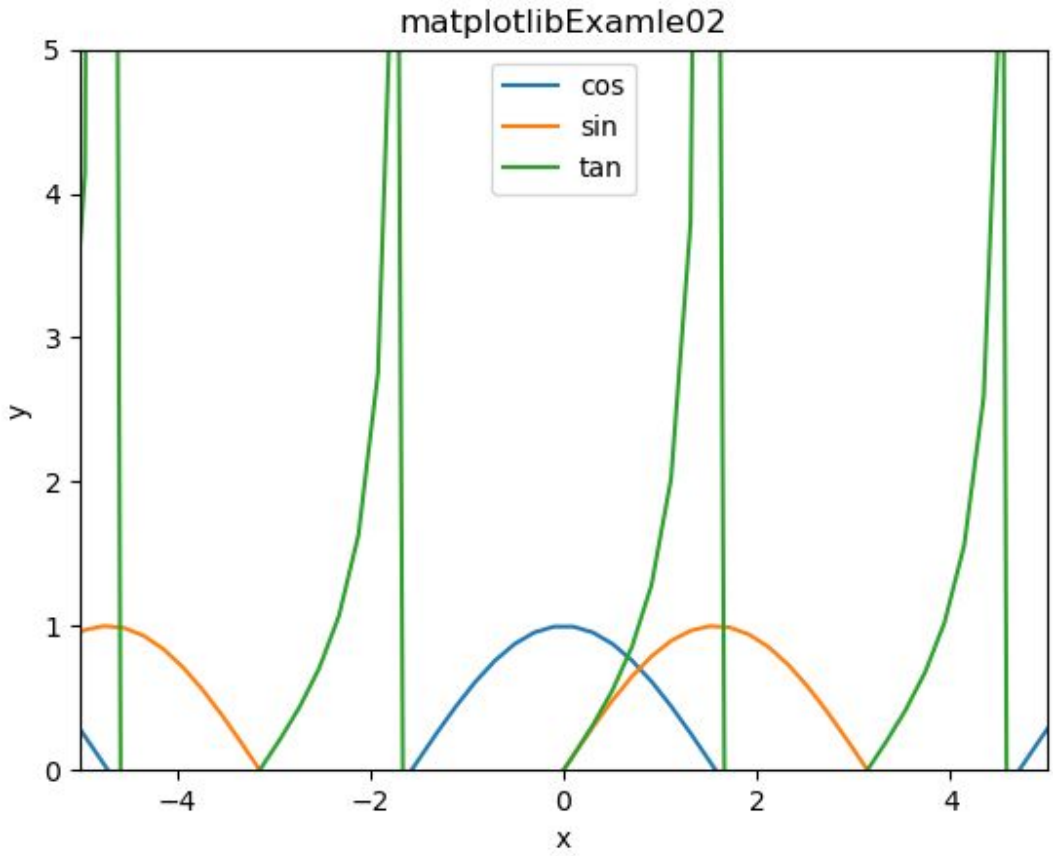
Matplotlib python plotting package (refer matplotlib example script **matplotlibExample.py**)

```
# plot a curve
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-10, 10, 100)
y=np.sin(x)
#create a new figure window
fig=plt.figure()
#plot on the figure
```

```
plt.plot(x, x, label='linear')
plt.plot(x, y, label='sin')

#set title and labels
plt.title('matplotlibExamle01')
plt.xlabel('x')
plt.ylabel('y')

#show legend
plt.legend()

# Show the plot
plt.show()
# save the fig
plt.savefig('matplotlibExamle01.png')
```



```
# limit the x and y axis
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-10, 10, 100)
y=np.sin(x)
#create a new figure window
fig=plt.figure()

#plot on the figure
plt.plot(x, np.cos(x), label='cos')
plt.plot(x, y, label='sin')
plt.plot(x, np.tan(x), label='tan')
#set title and labels
plt.title('matplotlibExamle02')

plt.xlabel('x')
plt.ylabel('y')

#limit the axis
plt.xlim([-5,5])
plt.ylim([0,5])

#show legend
plt.legend()

# Show the plot
plt.show()
# save the fig
plt.savefig('matplotlibExamle02.png')
```




```
#creating multiple subplot
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-10, 10, 100)
y=np.sin(x)
#create a new figure window
fig=plt.figure()

#plot on the figure
plt.subplot(1,3,1) # plt.subplot(rows,cols, id)
plt.plot(x, np.cos(x), label='cos')
plt.subplot(1,3,2)
plt.plot(x, y, label='sin')
plt.subplot(1,3,3)
plt.plot(x, np.tan(x), label='tan')
#set title and labels
plt.title('matplotlibExamle03')
```

```
plt.xlabel('x')
plt.ylabel('y')

#limit the axis
plt.xlim([-5,5])
plt.ylim([0,5])

#show legend
plt.legend()

# Show the plot
plt.show()
# save the fig
plt.savefig('matplotlibExamle03.png')
```