Python:

- **Variables**
  - Basically, it is a symbol that stores a value/data that can be used later on.
  - I.e, X = 5
  - Here, x is a variable that stores the value 5.
  - Storing data to a variable is sometimes called assigning data to a variable also.
  - Variables can not start with numbers like : 10, 10ba, 3c, etc. It should be in form of characters(number/Char/symbol)*.
  - So, when this 2 or any values are assigned to a variable then python automatically defines it's type, here 2 is of integer (int) type.
  - That's why python is called a dynamically typed language.
  - You can also do multiple assignment, like: a, b = 10, 6.7
  - Here 6.7 is of another data type that is called float.
  - There are more data types like int, float, string, etc. (will talk about it in a data type topic).
  - To delete a variable from the memory we use del (variable name), for example: del X. now there is no existence of X in the memory.
  - *%whos* is a command to check all the variables in the memory.

- **Operators**
  - Basic operators :

Symbols	Task Performed
+	Addition
-	Subtraction
/	Division (With decimal)
*	Multiplication
%	Module (Mod) (its gives reminder)
//	Floor division
**	To the power of

  - Bool types and comparisons :

Symbols	Task Performed
==	True, if it is equal
!=	True, if it is not equal
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
AND	If both are true then it will be true.
OR	Either one is true.
NOT	Change the value true to false.

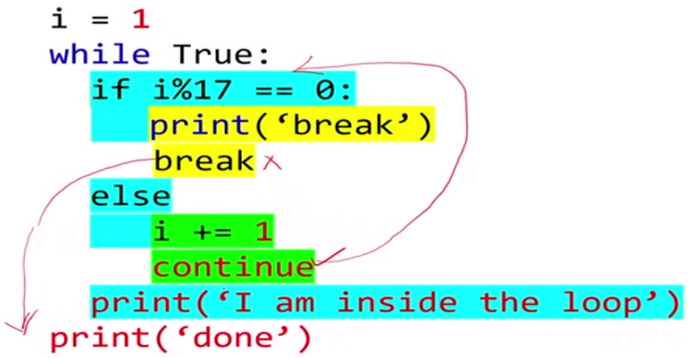
- **Some useful functions**

- round()
  - This function rounds the input values to a specified number of places or to the nearest integer.
  - For ex.: print(round(5.234)), output will be 5.
  - print(round(5.632)), output will be 6.
  - print(round(5.223423, 3)), output will be 5.223.
- divmod(x, y)
  - This function outputs the quotient and the remainder in a tuple.
  - For example, : divmod(27, 5), output will be (5, 2).
    - Here 5 is quotient and 2 is the remainder.
- isinstance()
  - This returns True, if the first argument is an instance of that class.
  - Multiple classes can also be checked at once.
  - For ex. : print(isinstance(1, int)), output will be True.
  - print(isinstance(1.0, int)), output will be False.
  - print(isinstance(3.4, (float, int))), output will be True.
    - Here it will check if 3.4 is of type float or int if either is true it will return true.
    - We can add more elements to it.
- pow(x, y, z)
  - Here x rises to the power y and remainder by z.
  - For example, : pow(2, 3), output will be 8, i.e: 2^3.
  - pow(x, y, z), output will be (x^y%z).
- input()
  - This function is used to take the input from the user.
  - For ex.: a = input("Enter your name/ something").
    - From here the user will get output as: *Enter your name/ something* and then the user can type anything.
  - Here the entered value or data will be of string type even if you enter a digit.
  - To convert it into integer we can type cast it.

- **Control Flow (Condition)**
  - If condition
    - This condition tells that if this is true then perform the given task.
    - That is, if (a > b): then print("a is greater than b").
    - Here if this is not true then it will not print/ execute the statement inside the if condition.
    - Here also comes *else* condition
  - If-Else condition
    - This condition tells that if this is true then perform the task given or if this is false then perform the task given in the else condition.
    - That is, if (a > b): print("a is greater than b") else: print("b is greater than a")
    - Let's assume there are more than one if conditions, here comes *elif*.
  - If-Elif-Else condition
    - This condition tells that if this is true then perform this task, else check another condition if this is true then perform this task, else check others conditions if there, at last if this is true then perform the else condition.
    - That is, if (a > b): print("a is greater than b") elif (a == b): print("a and b are equal") else: print("b is greater than a").
    - We can also add more elif conditions.
  - One line If-Elif-Else condition (shorthand if)
    - print("a") if a > b else print("equal") if a == b else print("b").
  - Nested If
    - This is nothing, just a condition inside a condition.

        - That is, if x > n: if x < m: print(“x”)
- **Comment**
    - This is basically a text note that gives an explanation about the source code, it is written in a program but ignored by compilers.
    - Single line comment
        - We use # for the single line comments
        - #This is a single line comment.
    - Multi-line comments
        - We use “”” “””for the multi line comments.
        - “””” This is a multi line comment…….
                …..also the comments. “””

- **Loops**
    - A programming element that repeats a portion of a set number of times until the desired process is complete.
    - Like I want to print “Hello” 5 times.
    - So here we can use a loop.
    - For Loop
        - For example: for i in range(5): print(“Hello”)
        - Here the range function will be from 0 to 4, 5 will be not included.
    - While Loop
        - In this loop it says that while this condition is true, execute the statement.
        - For example: i = 0 while(i < 5): print(“Hello”) i++.
- **Pass**
    - It just says do nothing.
    - We write this to make code readable else it does nothing.
- **Break**
    - Whenever this keyword comes it just says that just just stop the loop and come out of it.
- **Continue**
    - Whenever this keyword comes it just says to continue the loop regardless of whatever statement left after this keyword it skips it.

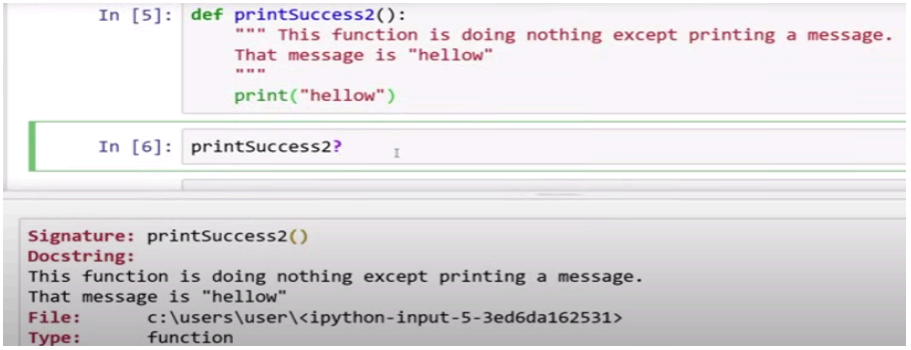


- **Else in for loop**

```
S = {"apple",4.9,"cherry"}
i = 1
for x in S:
    print(x)
    i+=1
    if i==3:
        break
    else:
        pass
else:
    print("Loop terminates with success")
print("Out side the loop")
```

- Here due to the break condition *"Loop terminates with success."* will not execute.

- **Function**
  - It is a block of code which can be reused multiple times.
  - It will run only when it is called.
  - You can also pass data, known as parameters.
  - Function can return data as a result.
  - Parameter and Argument
    - A *parameter* is the *variable listed inside* the parentheses in the *function definition*.
    - An *argument* is the *value that is sent* to the function when it is called.
  - Doc String
    - It basically gives the description about the function that you have given to it.
    - We can add a doc string just after declaration of the function like a comment so that if user type (function_name)? This will give you the description that you have written.
    - help(function_name) this will also give you the same output.



  - Variable number of input arguments
    - Let's assume you want to add n numbers of elements.
    - You don't know the value of n.
    - So we do, def add(*args): sum = 0 for i in range(len(args)): sum += argos[i] return sum.
    - Here we can also use (**args) and use it as a dictionary.

```
def myAddUniversal(*args):
    s = 0
    for i in range(len(args)):
        s += args[i] # s = s+args[i]
    return s
```

```
print(myAddUniversal(2,4,5,4.6,78))
```

```
93.6
```

  - Default value
    - We can also pass default value in the parameters that if the user doesn't give any value then it will use default value.
    - For ex.: def sum(a, b = 0): return (a+b).

- **isinstance()**
  - The isinstance() function returns True if the specified object is of the specified type, otherwise False.
  - For ex.: isinstance(msg, str) -> To check string type or not.
  - isinstance(msg, int) -> To check integer type or not.
- **Modules**
  - A file containing a set of functions you want to include in your application.
  - To use functions of that module you just have to import (module_name).
  - You can re-name your module by just writing import (module_name) as (new_name).
  - For ex.: import myModule as mx, now I can use the function with just mx.(function_name)().
  - In python there are several built in modules too, like platform.

- We can write the path if it is in a different folder.

```
import sys
sys.path.append('C:/ABC/')
```

```
import my_universal_functions as myfs
```

```
myfs.addAllNumerics??
```

- **String**
  - Sequences of characters are known as strings.
  - We use either single quotes or double quotes to write strings.
  - Space is also a character.
  - To write multi-line strings we use three double quotes or three single quotes.
  - Sting is a data type that is immutable that once it is assigned it can not change.
  - To do so we have to create another string.
  - Indexing and Slicing
    - We can access a character of a string by indexing.
    - Let's say a = "Hello_World", here the index of H is 0, e is 1, l is 3 and so on.
    - Now to access we type a[0] this will give H, same a[3] will give l.
    - Let's say you want just Hello, for this we type a[0:5]
    - So basically it prints the character from 0 to 4 (5-1), we can also add steps.
    - Like a[0:8:2] this will print HloW.
  - Negative Indexing
    - So if you want to access from the end of the string we use this.
    - Like a = Hello_World, a[-1] means d, a[-2] means l, and so on.
    - We can use this same for slicing too.
  - We can find the length of a string using len() function.
  - Methods of string
    - b = " A lot of spaces at begINNinG and End "
    - a = b.strip(), erase the spaces from beginning and end.
    - a.lower(), convert the string into a smaller case(lower case).
    - a.upper(), convert the string into upper case.
    - a.replace(" ", ","), this will replace every space with comma(,).
    - a.split(" "), this will split the string from space and make every one token into an element of the list.
    - a.capitalize(), it changes the first character of the string into upper case and others into lower case.
  - Skip sequence
    - It is just like, whenever you use "\" the next character is treated as a character.
    - \n is for a new line.
    - \t for a tab space.
- Data Structures

•List	Ordered,	changeable,	duplicates
•Tuple	Ordered,	unchangeable,	duplicates
•Set	Unordered,	addable/removable	no duplicates
•Dictionary	Unordered,	changeable,	no duplicate

List	Tuple	Set	Dictionary
L= [12, "banana", 5.3]	T= (12, "banana", 5.3)	S= {12, "banana", 5.3}	D={"Val":12,"name":"Ban"}
L[1]	T[2]	X in S	D["Val"]
L = L + ["game"] L[2] = "orange"	Immutable T3 = T1+T2	S.add("new Item") S.update({"more","items"})	D["Val"] = newValue D["newkey"] = "newVal"
del L[1] del L	Immutable del T	S.Remove("banana") del S	del D["Val"] del D
L2 = L.copy()	T2 = T	S2 = S.copy()	D2 = D.copy()
....	....	....	....

- **Numpy**
  - Why Numpy?
    - Numpy is Faster.
    - It is fast because of its contiguous memory allocation.
  - We have to import it to use it.
  - Import numpy as np
    - A = np.array([1, 2, 3, 4, 5])
    - B = np.array((1, 2, 5, 6, 8))
    - Here array is a function of numpy and we can use it as a list or a tuple.

```
import numpy as np
```

```
a = np.array([1, 3, 4, 5, 6], dtype='i')
```

```
b = np.array((2, 4, 6, 8), dtype = 'f')
```

```
print(a)
print(b)
```

```
[1 3 4 5 6]
[2. 4. 6. 8.]
```

    - Here 'i' and 'f' are integer and float types used to declare that the array is of particular type.
  - Dimensions
    - A = np.array([[1, 2, 3], [4, 5, 6]])
    - print(a.ndim)
      - Here it will print the dimension of array, i.e. 2
    - Now to access let's say 3, then we have to write a[0, 2].
      - Where '0' is for the first list of the array and 2 is for the element of that list.
    - **NOTE**: Length of every list or tuple or whatsoever have to be the same.

```
d = np.array([[[1, 3, 4],[2, 4, 6],[55, 33, 22]],
              [[-1, -3, -4],[-2, -4, -6],[-55, -33, -22]]])
```

```
print(d[1,2,2])
```

```
-22
```

    - Here this is a 3 dimensional array.
    - We can make n-dimensional array.
    - Shape

- It tells how many 1-d arrays are and how many 2-d arrays are and how many 3-d arrays are and so on.

```
d = np.array([[[1, 3, 4],[2, 4, 6],[55, 33, 22]],
              [[-1, -3, -4],[-2, -4, -6],[-55, -33, -22]]])
```

```
print(d.shape[0])
print(d.shape[1])
print(d.shape[2])
```

```
2
3
3
```

```
d.shape
```

```
(2, 3, 3)
```

- D.shape gives a tuple and index 0 tells no. of 1-d, index 1 tells no. of 2-d and so on.

■ Size
- D.size, this will give you the total number of elements in the whole array.
- For ex. : in the above case it will print 18.

■ Nbytes
- D.nbytes, this will give you the total number of space it is taking in the memory.
- For the above case it is taking 72 bits of space.

■ There are more functions we can explore by clicking d.(tab).

○ Arange
■ This np.arange(100).
■ It will make an array that starts from 0 and to till 99.
■ We can use a starting point and step point just like we do in a range function.
■ Like np.arange(1,101,2), here starts from 1 till 100 by step 2.

NOTE:
Uniform mean:
Kind of straight parallel line.

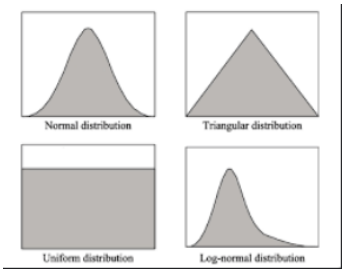
Normal mean:
Kind of a hill shaped.

○ Random
■ *Random.permutation*
- This function just shuffles the elements.
- For ex.: np.random.permutation(np.arange(11))
- It will give a list from 0 to 10 but the elements will be in shuffled.

■ *Random.randint*
- This function will give to a random integer between the range
- For example: np.random.randint(0,101), this will give you a random integer from 1 to 100.

```
f = np.random.randint(1, 101, 3)
print(f)
```

```
[95 20 91]
```



- np.random.rand(4), give 4 random numbers between 0 and 1, numbers will be uniform.
- np.random.randn(4), give 4 random numbers between 0 and 1, numbers will be normal.

■ There are so many functions in random().
■ np.zeros(3), creates an array of length 3 with all zero.
■ np.ones(3), creates an array of length 3 with all one.
■ reshape
- This just reshapes into a desired matrix.
- Like np.arange(8).reshape(2,4)
- This is a 2 by 4 matrix with numbers 0 to 7.

○ Slicing
■ We can use slicing like we used to do in lists.

- Masking
  -
- Broadcasting
  - Basically, scalar multiplication of a matrix.
- Np.hstack
  - This will add another matrix horizontally.
- Np.vstack
  - This will add another matrix vertically.
- Np.sort
  - This will sort the array.
- How fast is np's functions are:

```
B = np.random.rand(1000000)
%timeit sum(B)
%timeit np.sum(B)  # B.sum()
```

```
307 ms ± 22.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
2.77 ms ± 260 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Pandas

- It is kind of a dictionary but not exactly.
- We have to import pandas to use it.
- Like import pandas as pd.
- Now, a = pd.Series([1, 2, 3, 4, 5], index = ['a', 'b', 'c', 'd', 'e'])
- Here the series[1, 2, 3, 4, 5] is a numpy array and its index is ['a', 'b', 'c', 'd', 'e'].
- We can access any element by its index like a['c'] will give 3 and a['e'] will give 5.
- We can also use slicing like a['a':'c'] this will give [1, 2, 3], here you can notice this slicing will also include the ending index.
- We can convert dictionary into pd.Series, m = {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f'}
- Now by pnd = pd.Series(m), this is now a panda series.
- DataFrame
  - With this we can create a table.
  - Like:

```
grades = {'f':0.9, 'b':5, 'c':5.6, 'd':5.4, 'e':6.9}
grads = pd.Series(grades)
```

```
marks = {'a':90, 'b':59, 'c':69, 'd':85, 'e':87}
mark = pd.Series(marks)
```

```
df = pd.DataFrame({"Grades":grads, "Marks":mark})
print(df)
```

```
   Grades  Marks
a     NaN   90.0
b     5.0   59.0
c     5.6   69.0
d     5.4   85.0
e     6.9   87.0
f     0.9    NaN
```

    - Here NAN stands for "Not A Number".
  - What if you want to add a column to this table?
    - So, to add column : df(table_name)[New_column_name] = then items.
    - df["grade"] = round((df['Marks']/9.5), 2)
    - This is how you can add any column
  - What if you want to delete any column

        - To do so we can just type del df[“column_name”]
        - Now the column will get deleted.
    - Masking
        - This means getting only certain items that satisfy the conditions.

```
print(df[df["grade"]>8])
```

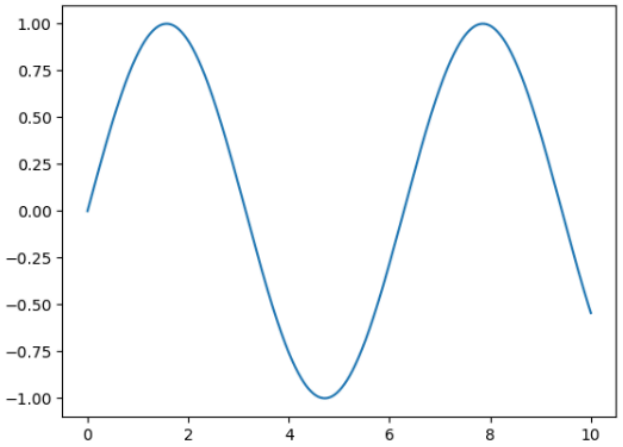
```
   Grades  grade
a     NaN   9.47
d     5.4   8.95
e     6.9   9.16
```

    - NAN
        - Table gives NAN if there is no value for it.
        - If you don’t want it then you can use a.fillna(0), this will plot 0 instead of every NAN.
        - Or you can do a.dropna(), this will drop all the records with NAN.
    - Indexing
        - *Explicit index(loc)*
            - If you want to use the index that you provide in the pd.series.
            - Then you will use a.loc[a:b], here a,b are the index that you have provided.
        - *Implicit index(iloc)*
            - If you call normally a[b:c], this will use normal indexing.
            - Or you can also use it as a.iloc[b:c].
            - The outputs of the above will be the same.
- Drop
    - df.drop([‘sno’, ‘lastUpdate’], axis=1, inplace=True])
    - Here ‘sno’ and lastUpdate will be deleted.
    - axis=1
        - This calls to drop the whole column.
        - If we write 0 instead of 1, then it will only drop ‘sno’ and ‘lastUpdate’.
    - inplace=True
        - This tells it to update the file right away.
        - Don't create a new temporary file.
- Rename
    - df.rename(Columns={‘observationDate’:’Date’, ‘Province/State’:’State’}, inplace=True).
    - Columns={‘observationDate’:’Date’, ‘Province/State’:’State’}
        - This will rename the observationDate to ’Date’ and Province/State to ’State’.
    - inplace=True
        - This tells it to update the file right away.
        - Don't create a new temporary file.
- Date format
    - To change date format
    - df[‘Date’] = pd.to_datetime(df[‘Date’])
- Head
    - df.head(10)
    - This will only give the 10 records from the top.
- Description()
- info()
- groupby().sum().set_index()

- **Matplotlib**
  - To use this module we have to import it.
  - Pyplot
    - Import matplotlib.pyplot as plt.
    - X = np.linspace(0, 10, 1000)
      - This will give you 1000 points between 1 to 10.
    - Y = np.sin(x)
    - *Plot*
      - plt.plot(x,y)

```
x = np.linspace(0, 10, 1000)
y = np.sin(x)
print(plt.plot(x, y))
```

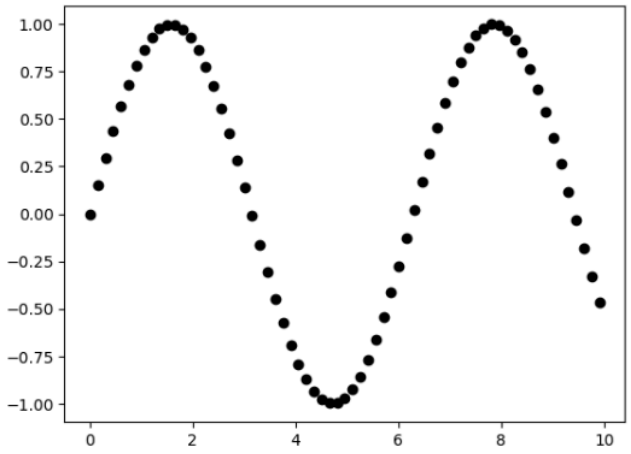
```
[<matplotlib.lines.Line2D object at 0x000001FF562BA4D0>]
```



    - *Scatter*
      - plt.scatter(x[::15], y[::15], color='red')

```
print(plt.scatter(x[::15], y[::15], color="black"))
```

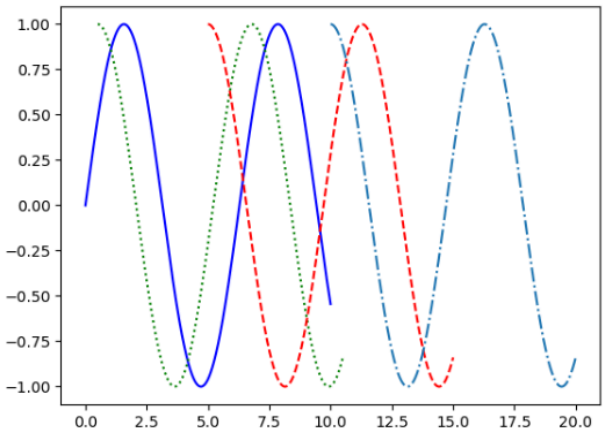
```
<matplotlib.collections.PathCollection object at 0x000001FF56A9BF50>
```



    - You can also plot more than one curve in a single graph.

```
plt.plot(x,y,'-b')
plt.plot(x+0.5, np.cos(x),':g')
plt.plot(x+5, np.cos(x),'--r')
plt.plot(x+10, np.cos(x),'-.')
```

```
[<matplotlib.lines.Line2D at 0x1ff5c907250>]
```



- In the above we can change color and the pattern.
- For color we use color attributes.
- And for pattern:
  - '-g': solid green
  - '--c': dashed cyan
  - '-.k': dashdot black
  - ':r': dotted red
  - And more on.

- **SciPy**
  - It is an open source python library used to solve mathematical and scientific problems.
  - It stands for Scientific Python.
  - It is built on the NumPy extension and allows users to manipulate and visualize data with a wide range of high level commands.
  - So if you import scipy then you don't need to import numpy.
  - Sub- packages in SciPy

Name	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions

  - Clustering

- Clustering is the task of dividing the population or data points into a number of groups that data points in the same groups are more similar to other data points in that same group and dissimilar to the data points in other groups.
- Each group which is formed from clustering is known as a cluster.
- There are two types of clusters:
    - Central
    - Hierarchy