**Jupyter Notebook**

### Jupyter Notebook

* Reporting Tool

1. GUI

2. Syntax Rule

3. Basic Components

- Operators

- Data types

- Conditional statements and

- Loops

#### Step 1: Launch Jupyter

* Open Anaconda Navigator >> Launch Jupyter Notebook for Python
* Open Anaconda Prompt/or regular cmd promplt/Terminal >> type Jupyter notebook

#### Step 2: Open Jupyter Notebook

#### Step 3: Rename and saving a notebook

* Saved as ipynb

### Components of Jupyter Notebook

1. Cells - Area where code/text/information can be written, processed and displayed
2. States:
   * + Select: Its non-editable and all shortcuts will work in this state (Blue) - Esc key
     + Edit: Write the text and the codes in the jupyter notebook (Green) - Click inside the cell
3. Modes & Shortcut

Code: Syntax correct python code

Code Esc + Y

Raw NBConvert - Raw text mode Esc + R

Markdown Mode - Stylized text Esc + M

Insert a cell below Esc + B

Insert cell above Esc + A

Paste the cell Esc + V

Delete the cell Esc + DD

Copy the cell Esc + C

Cut the cell Esc + X

Paste the cell Esc

Execute the code/text

Ctrl + Enter - Execute the code and stay there

Shift + Enter - Execute the code and go to next cell

* + 1. Text: Write raw text/plain text (raw NBConvert)
    2. Markdown: Advanced Text Mode where we can write stylized text (HTML tags converted into Symbols)

### Syntax Rules:

1. Python is case sensitive
2. Loosely/dynamically typed language
3. [#] symbol in code mode is used for comment lines.
4. Follow INC rules for naming (Identifier naming conventions)
   * Purely alphanumeric
   * No Symbols/special characters allowed except underscore.
   * Names should start with characters or underscore.
   * Reserved words/keywords should not be used (All keywords appear in green color)

**Basic components of python:**

1. Operators

2. Data types

3. Conditional statements

4. Loops

5. Data structures

### Operators:

1. Arithmetic **/ \* + - // %**
2. Assignment **=**
3. Assignment + Arithmetic **+= -= \*= /= %=**
4. Relational **> >= < <= == != <>[**Output will be **T**rue/**F**alse]
5. Logical **and or not**
6. Membership **in not in**

Note:

* Single = is for assignment & == is for comparison
* Not Equal to != &<>
* **T**rue/**F**alse is different from **t**rue/**f**alse

### Data Types:

* int (-2147483648 to +2147483648)
* float
* Str
* long
* bool
* date

1. Char
   1. str
2. Numeric
   1. whole numbers int/long
   2. floating point numbers/decimals float
   3. bool
   4. dates
3. str
   1. float -inf to + inf
   2. long -infinity to + infinity only intiger values
   3. int (-2147483648 to +2147483648)
   4. bool 1/0 or true/false

* str()
* float()
* long()
* int()
* bool()

**INC (identifier naming convention) rules for python**

1. All the identifiers should be purely alphanumeric

2. No special char is allowed except underscore\_

3. Identifier names can start with alphabets or the underscore

4. Don't make use of any reserved keywords as identifier names

**Conditional statements**

if condition/s:

operation/s

elif condition/s:

.

.

.

else

operations

main() {

statement1;

statement1;

if () (

statement 3;

);

}

**Indentation code (recommended way to write code in python)**

main():

statement1

statement1

if ():

statement 3

print (x)

### Loops: Python supports two types of loops

- Conditional loops

- Ranged loops

#### Conditional loop (while loop)

* while loop iterates for the True condition/s and it comes out of the loop when given logical condition/s become False
* If the condition is true, the block of code under it is executed
* Be careful while using a while loop because if no termination condition is provided; the loop will run infinitely

#### Ranged loop (for loop)

* For loops have fixed range in which they can iterate.

**Syntax:**

for x in [range]:

Operations

* In order to use for loops, we need some range and these ranges can be created using a range function

# range() function is used to generate the range of numbers

# range() takes three arguments - range(start, end, inc)

# It returns a "list" of numbers starting from start till end - 1 in steps of inc value

# Default value of increment is 1

x1 = range(0, 10)

x2 = range(0, 10, 2) # third argument is for step size

x3 = range(10, 0, -1)

# for loop uses range function for iteration

for x in range(1, 11):

print (x)

#### 1. Loops

#### Loops: Python supports two types of loops

**- Conditional loop**

#### Conditional loop (while loop)

* while loop iterates for the True condition/s and it comes out of the loop when given logical condition/s become False
* If the condition is true, the block of code under it is executed
* Be careful while using a while loop, because if no termination condition is provided; the loop will run infinitely

**- Ranged loop**

#### Ranged loop (for loop)

* For loops have fixed range in which they can iterate.

**Syntax:**

for x in [range]:

**Operations**

* In order to use for loops, we need some range and these ranges can be created using a range function
* range() function is used to generate the range of numbers
* range() takes three arguments - range(start, end, inc)
* It returns a "list" of numbers starting from start till end - 1 in steps of inc value
* Default value of increment is 1

x1 = range(0, 10)

x2 = range(0, 10, 2) # third argument is for step size

x3 = range(10, 0, -1)

* for loop uses range function for iteration

for x in range(1, 11):

print (x)

* When we talk about any object; two things we need to answer

1. What is the type of the data - Data Types
2. How is the information stored - Data Structures

### 2. Data Structures

A data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

They can be classified into two types based on the content they can have:

1. Homogeneous: All data points of same datatype

2. Heterogeneous: All data points in the data structure are not of same data type.

All base data structures available in python are 1D and heterogeneous. **Types of DS available in python**:

1. Tuple

Tuples: Created using () brackets or even without brackets

2. List

List: Created using [] brackets

* Both **Tuple** and **List** are accessed using [] brackets; where these square brackets will take one or more numbers as input
  + These numbers are Indexes from range 0 till length - 1
  + **colon :** is used to create the range
  + We may use range function to generate the range of numbers
  + **len()** function gets the len of the Tuple or List
  + **Type()** function to get type of data structure
* *Tuples are* ***immutable*** *and Lists are* ***mutable****.*Apart from this; there is no other difference in two data structures.
* Values inside a list can be reassigned Valid: l1[2] = 99
* values in the tuple are fixed and can't be changed Invalid: t1[2] = 99
* item reassignment in **tuples** and **lists**

l1[3] = 99

t1[4] = 99

* Functions associated with tuples or list
* **dir():** Listing function (By default, displays everything that is present in current session)

**Functions of Tuple Object**

print (dir(t1))

t1 = 12, 23, 34, 23, 21, 34

* + count(t1) # Invalid: Not correct way of accessing functions of the object
  + count function will count how many times a value appears in the tuple t1.count(23)
  + index function will return the index/position of first occurrence of the given value t1.index(34)

#### Functions of list object

print(dir(l1))

* count: return number of occurrences of value << **count(value)**>>
* index: return the index/position of first occurance of the given value << **index(value)** >>
* append: append object to end << **append(value)** >>
* insert: insert object before index << **index(pos, value)**>>
* extend: extend list by appending elements from the iterable << **extend(iterable)** >>
* pop: remove and return item at index (default last) << **pop()** >> or << **pop(pos)** >>
* remove: remove first occurrence of value. << **remove(value)** >>
* reverse: reverse the position of the values *IN PLACE*
* sort: arrange the values in asc or desc << **sort(reverse = False)** >>

#### To get help on python

1. Use help() function

2. use ?

**3. Dictionary**

#### While discussing these data structures, we will focus on following:

* How to Create
* How to Access
* Apply condition
* Functions associated with them

**Dictionary: Created using {} brackets**

#### Rules for dictionary DS

1. Key should be unique.
2. A key must be simple i.e. str/int/float
3. A value can be ANYTHING!
4. To access the values in dictionary, we can't use index or position values.
5. Get the values from dict using the keys
6. Key & Values should be separated by **: (colon)**

**Functions**

Set of instructions that can perform some set task and may or may not return a value - They are reffered by a name

* **def ->** Keyword to create a UDF
* **return ->** will provide the output from the function
* **Arguments ->** Inputs given to a function.
* Positional
* Named
* Function is available in correct session

Syntax:

def FunctionName():

Set of operations

return(value)

def AddNo(x, y):

return(x + y)

Passing positional arguments - args

AddNo(3, 5)

Passing Named arguments/key-word arguments - kwargs

AddNo(y = 20, x = 30)

We can also give default values to the arguments in function defination

def AddNo\_Default(x = 0, y = 0):

return(x, y)

#### Argument overloading in functions:

* If we are not sure how many arguments we will pass to a function; we can use - args or kwargs instaed of giving argument names
* \*args will convert the values passed to a function into a tuple
* \*\*kwargs will convert values passed to a function into a dictionary (key-value pair)
* Only positional arguments accepted, no named arguments allowed

def MyFunc (args):

print (args)

* function for addition of n arguments

def MyFunc1(args):

Sum = 0

for i in args:

Sum += i

return (Sum)

* only named arguments accepted, no positional arguments allowed

def Myfunc2(kwargs):

print(kwargs)

----------------------------------------------

Topics need more clarity,

* Count
* Extend
* Reverse
* Pop
* Remove
* **Dummy**
* Set

### User Defined Functions (UDFs)

Functions can declared using def

* def MyFunc1():

return(10)

* def MyFunc2(x, y):

return(x + y)

* def MyFunc3 (a, b):

Sum = a + b

Prod = a \* b

quo = a / b

rem = a % b

return ({'Sum': Sum, 'Prod': Prod, 'Quotient': quo, 'Rem': rem})

* def MyFunc (\*args): # \*args \*\*kwargs

Sum = 0

for x in args:

print (type(x))

if x % 2 == 0:

Sum += x

return (Sum)

### Lambda Function

## **Lambda** - Anonymous function

* Usually short functions, useful for simple calc and we don't have to create UDF
* Disadvantage - Unlike UDF, cannot be used multiple time

Write a UDF to get the square of the number

def fn\_sq(x):

return(x \*\* 2)

**Lambda**  equivalent for above UDF

lambda x: x \*\* 2

Lambda function to get sum of two numbers

lambda x, y: x + y

**Map function applies a lambda function to all the items in an iterable.**

list(map(fn\_sq, (3, 4)))

get the square of all items in the list

l1 = [1, 2, 3, 4, 5]

l1\_square = []

for i in l1:

l1\_square.append(i \*\* 2)

l1\_square

Map allows to implement this in a much simpler and nicer way.

Map(func, iterables)

l1 = [1, 2, 3, 4, 5] ## list iterable

l1\_square = list(map(lambda x: x \*\* 2, l1))

l1\_square

**Filters creates a list of elements for which a function returns true**

n\_list = range(-5, 5)

less\_than\_zero = list(filter(lambda x: x < 0, n\_list))

print(less\_than\_zero)

### OOP concepts

OOP - Object oriented Programming

* Encapsulation
* Inheritance
* Polymorphism
* Abstraction

------------------------------------------------------------------------------------------------------------------------------------------  
Numpy

* Importing numpy library
* Creating arrays and initializing
* Special initializing functions
* Slicing and indexing
* reshaping arrays
* Numpy Maths
* Combining arrays
* Basic algebraic operations using numpy arrays
* Compare the memory consumption of list and numpy array

import sys

**import numpy as np**

* All numerical and mathematical functions
  + logs, arithmetic calc
  + matrix operations
  + Stats functions
  + trigonometric
* Pandas makes use of numpy stats functions (stats measurements, stats tests) through inheritance(internally)
* To genearte some prob distributions, we use numpy (numpy.random.xxxxx())

**Creating numpy arrays and initializing**

From python default data structures

list, tuple

**Create one dimensional array**

array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

a1 = np.array([23, 44, 33, 66, 77, 88, 99, 12])

**Create a two dimensional array**

a2 = np.array([[23, 44, 33, 66], [77, 88, 99, 12]])