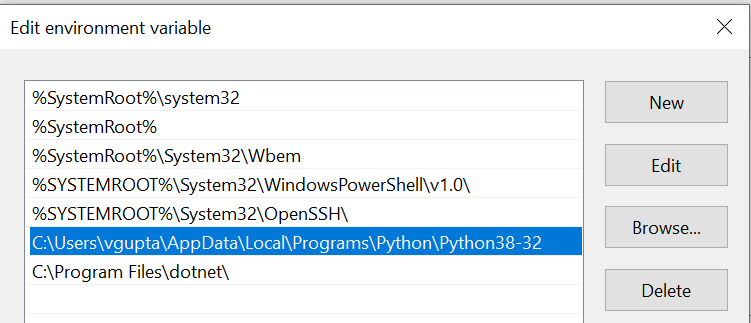
Python

# Getting Start

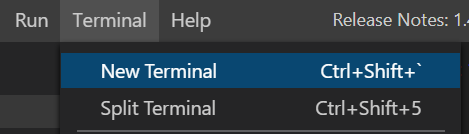
## Installing on Visual Studio Code

1. Download Visual Studio Code.
2. Install extension from the Visual Studio Marketplace. The Python extension is named Python and it's published by Microsoft.
3. Install a Python interpreter from <https://www.python.org/downloads/>. Latest version is 3.8.
4. Verify the Python installation
   1. Run py -3 –version on CMD.
   2. If it does not show set Python path on

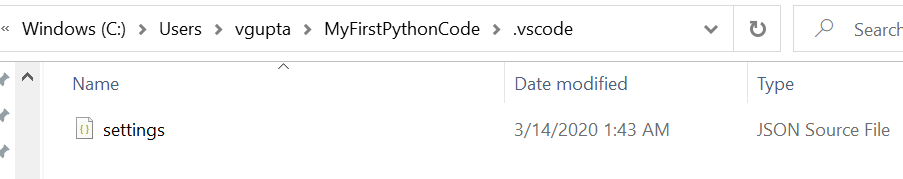


## Setting Up project in VS Code

1. Open new terminal in VS Code



1. Enter below mentioned command on windows cmd
   1. Cd\
   2. D:
   3. Cd D:/PythonProjects
   4. mkdir hello
   5. cd hello
   6. code .
2. By starting VS Code in a folder, that folder becomes your "workspace". VS Code stores settings that are specific to that workspace in **.vscode/settings.json**, which are separate from user settings that are stored globally.



1. Alternately, you can run VS Code through the operating system UI, then use **File > Open Folder** to open the project folder

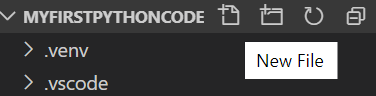
## Select a Python interpreter

From within VS Code, select a Python 3 interpreter by opening the **Command Palette** (Ctrl+Shift+P), start typing the **Python: Select Interpreter** command to search, then select the command. You can also use the **Select Python Environment** option on the Status Bar if available (it may already show a selected interpreter, too):

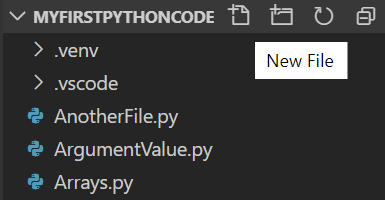
No interpreter selected

# Create a Python Hello World source code file

1. From the File Explorer toolbar, select the New File button on the MYFirstPythonCode folder:



1. Name the file <fileName>.py, and it automatically opens in the editor:



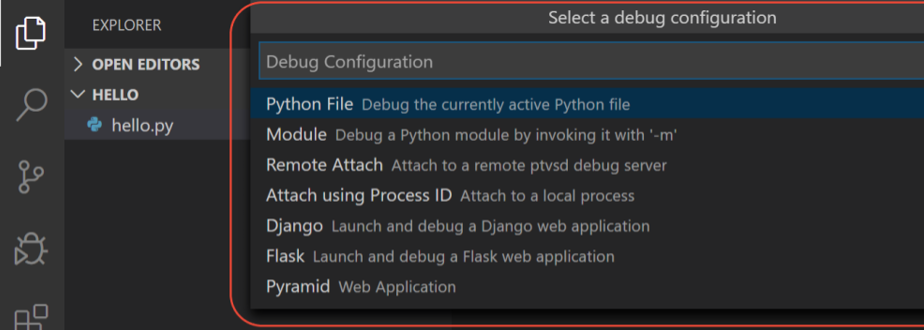
1. By using the .py file extension, you tell VS Code to interpret this file as a Python program, so that it evaluates the contents with the Python extension and the selected interpreter.
2. In this file you can type Python code.

## Run Python File

1. Right-click anywhere in the editor window and select **Run Python File in Terminal** (which saves the file automatically)
2. Select one or more lines, then press **Shift+Enter** or right-click and select **Run Selection/Line in Python Terminal**. This command is convenient for testing just a part of a file.
3. From the Command Palette (Ctrl+Shift+P), select the **Python: Start REPL** command to open a REPL terminal for the currently selected Python interpreter. In the REPL, you can then enter and run lines of code one at a time.

## Configure and run the debugger

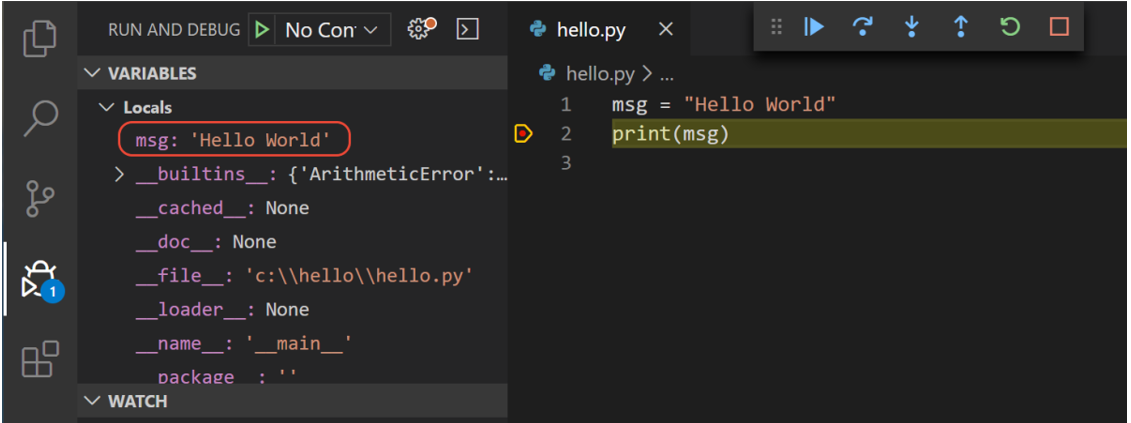
1. Place break point on line to be debugged.
2. Next, to initialize the debugger, press F5. Since this is your first time debugging this file, a configuration menu will open from the Command Palette allowing you to select the type of debug configuration you would like for the opened file.



1. Select **Python File,** There are various other configurations.
2. Left to right: continue (F5), step over (F10), step into (F11), step out (Shift+F11), restart (Ctrl+Shift+F5), and stop (Shift+F5).

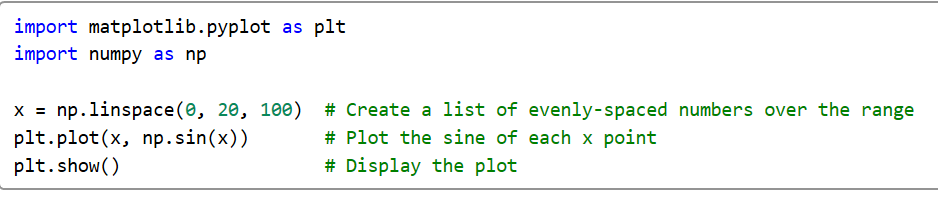


1. Local variables window.



## Install and use packages

1. create a new file called standardplot.py, and paste in the following source code:



1. To install the matplotlib package (which also installs numpy as a dependency), stop the debugger and use the Command Palette to run **Terminal: Create New Integrated Terminal** (Ctrl+Shift+`)). This command opens a command prompt for your selected interpreter.

**pip install matplotlib**

1. Create Virtual Environment. This avoid nstalling packages into a global interpreter environment, instead use a project-specific virtual environment that contains a copy of a global interpreter. Once you activate that environment, any packages you then install are isolated from other environments
   1. Create and activate the virtual environment: [When you create a new virtual environment, you should be prompted by VS Code to set it as the default for your workspace folder. If selected, the environment will automatically be activated when you open a new terminal.]

Enter follow code in terminal

py -3 -m venv .venv

.venv\scripts\activate

* 1. You might be required to change shell execution policy

Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope Process

* 1. Select your new environment by using the **Python: Select Interpreter** command from the **Command Palette**.

1. Install the packages

# Windows (may require elevation)

# Don't use with Anaconda distributions because they include matplotlib already.

python -m pip install matplotlib

1. Uninstall package

pip uninstall numpy

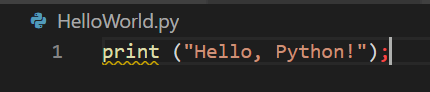
1. Once you are finished, type deactivate in the terminal window to deactivate the virtual environment.

# Python Tutorial

Python is interpreted, object oriented, high level, procedure oriented language

## Basic Syntax

Hello world code just Require one line



### Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers −

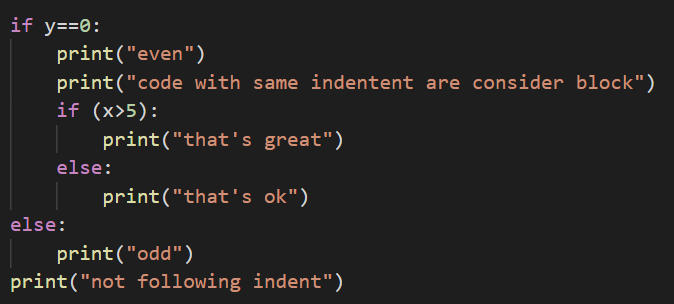
* Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
* Starting an identifier with a single leading underscore indicates that the identifier is private.
* Starting an identifier with two leading underscores indicates a strongly private identifier.
* If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

### Reserved Words

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

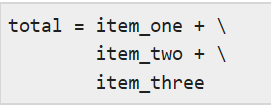
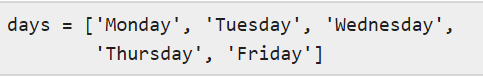
### Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced. The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.



### Multi-Line Statements

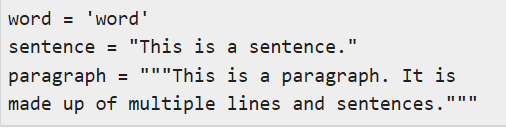
Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.

Statements contained within the [], {}, or () brackets do not need to use the line continuation character.  

### Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal –



### Comments in Python

A hash sign (#) that is not inside a string literal begins a comment

#### Documentation

Can be exported to xml, txt, html format and help full for others

“””

………………………..

………………………..

………………………..

“””

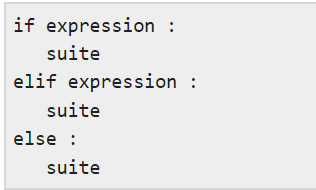
### Multiple Statements on a Single Line

The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts a new code block



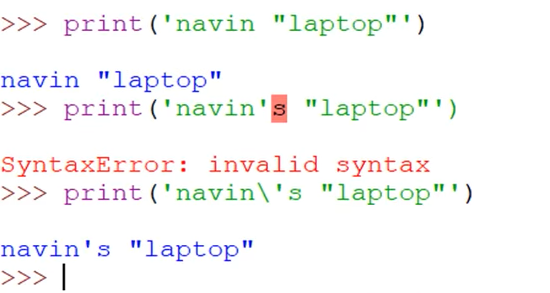
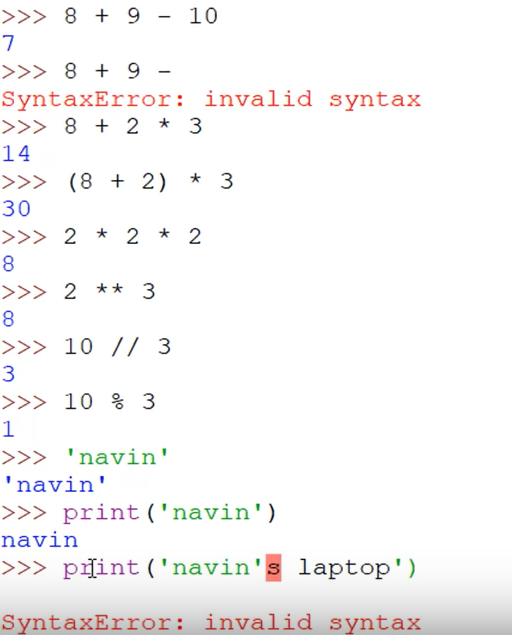
### Multiple Statement Groups as Suites

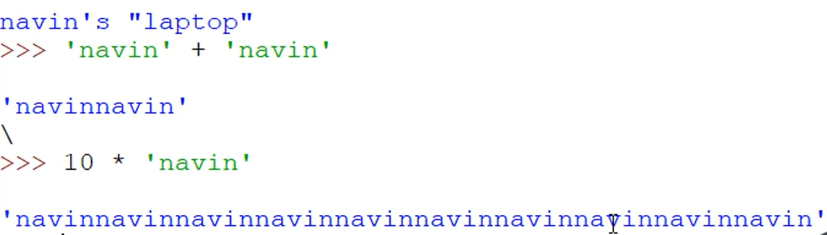
Compound or complex statements, such as if, while, def, and class require a header line and a suite. Header lines begin the statement (with the keyword) and terminate with a colon ( : )



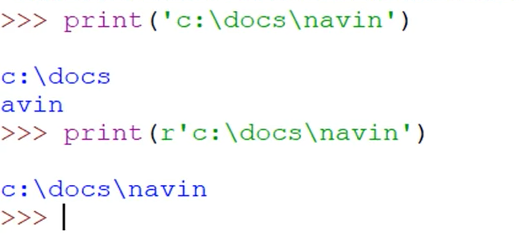
## Getting Started

Some legal statements in Python. We can directly type these normal English command and Python will give us output.





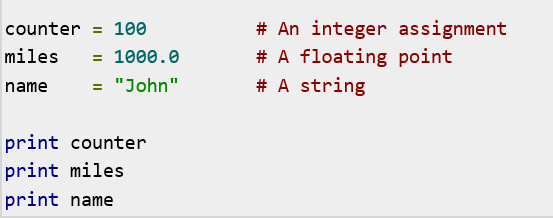
### Raw String



## Variable in Python

### Assigning Values to Variables

We do not need to define type of variable in Python. It auto detects type based on value.



### Multiple Assignment

We can assign multiple variables in single line.





### Standard Data Types

* None : equivalent to null
* Numeric
  + int (signed integers)
  + long (long integers, they can also be represented in octal and hexadecimal)
  + float (floating point real values)
  + complex (complex numbers) : a+bi
  + bool
* Sequence
  + String
  + List
  + Tuple
  + Set
  + Range
* Dictionary / Map

#### Python Numbers

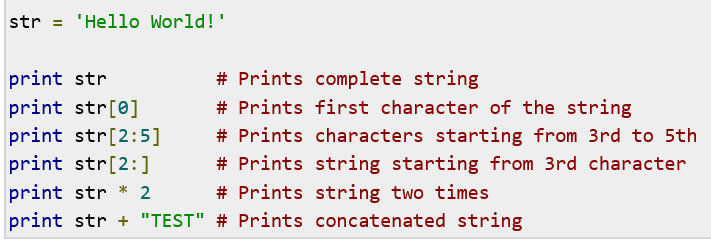
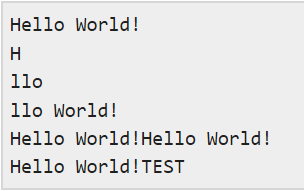


#### Strings

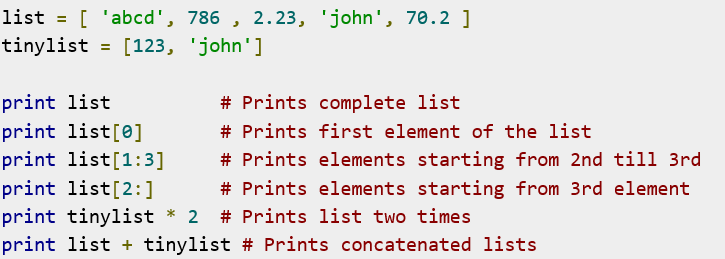
Items belonging to a list can be of different data type.

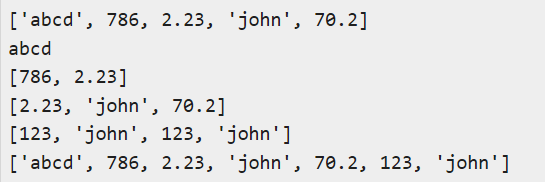
A list contains items separated by commas and enclosed within square brackets ([])

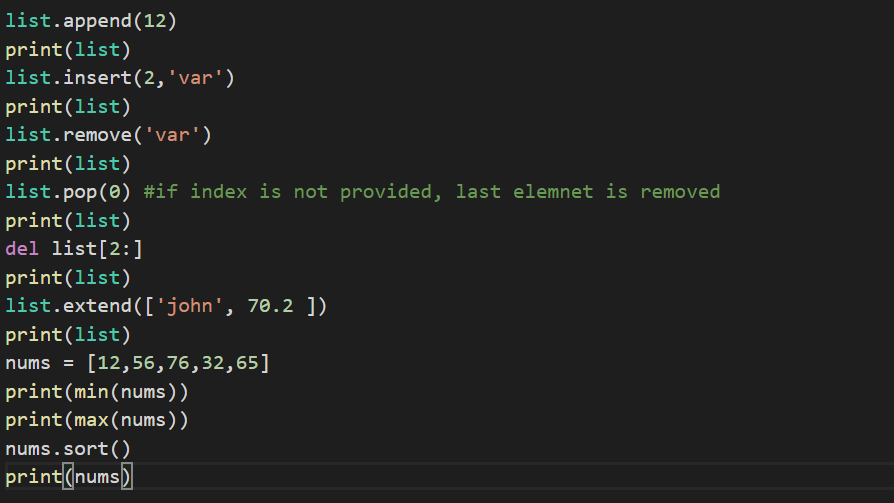
List is mutable. And its value can be changed.

#### List





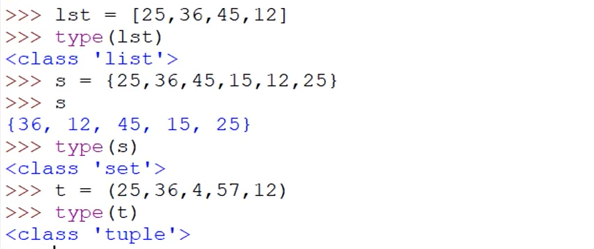




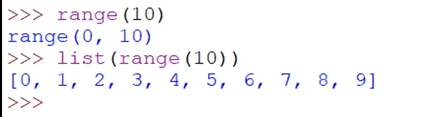
#### Tuple & Set in Python

Tuples are enclosed in parentheses ( ( ) ) and **cannot be updated**. Tuples is **read-only** lists. So iteration is fast.

Set is mutable, but it **does not follow sequence** like Lis. Defined by {}. Gets unique value. Indexing is not supported to.



#### Range

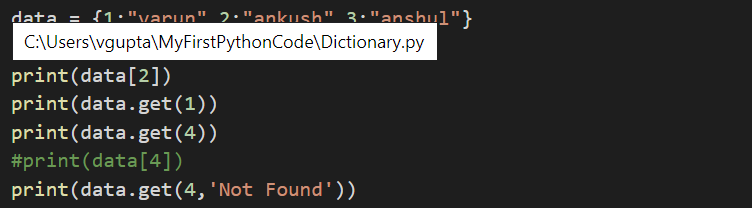


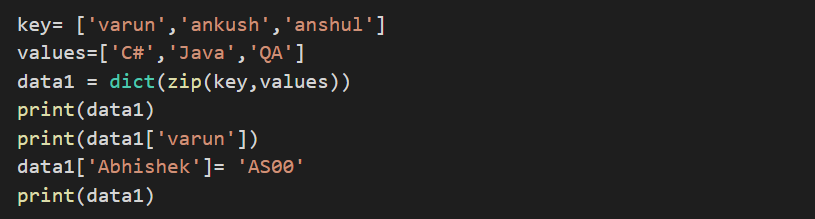
To get only even number range. First parameter states starting point, second state ending number, and third is difference to be considered in range.

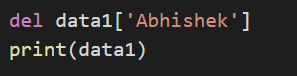


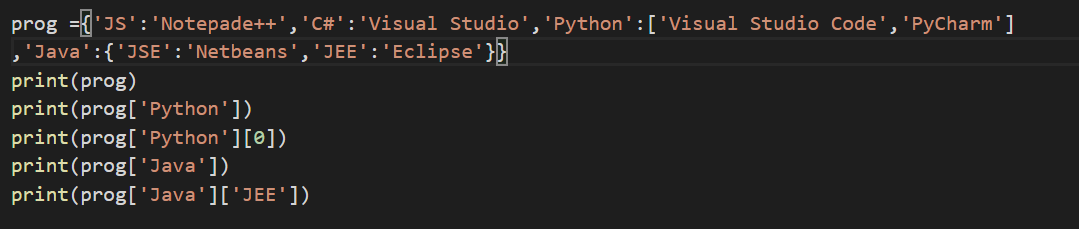
#### Python Dictionary

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([])



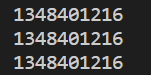
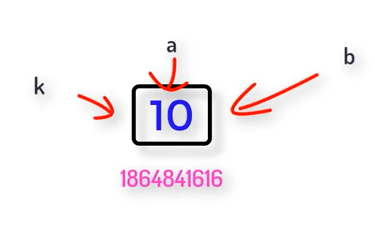


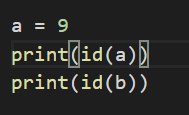
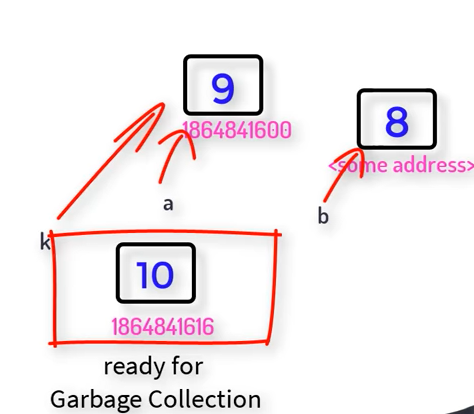




### ID of Variable

In python if we have multiple variable holding same value, there address will be same. Thus variable are called tags.

### Constants

We do not have concept of constants in Python. We can just show intent by defining variable in all CAPS.

PI = 3.14

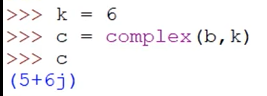
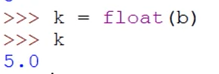
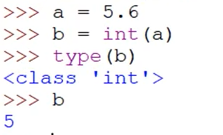
### Variable Types

print(type(PI))

output:

<class 'float'>

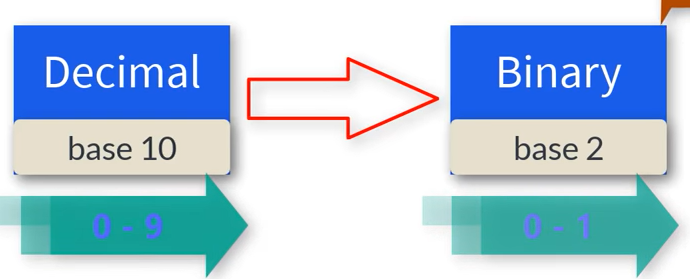
### Data Type Conversion



## Number System Conversion in Python

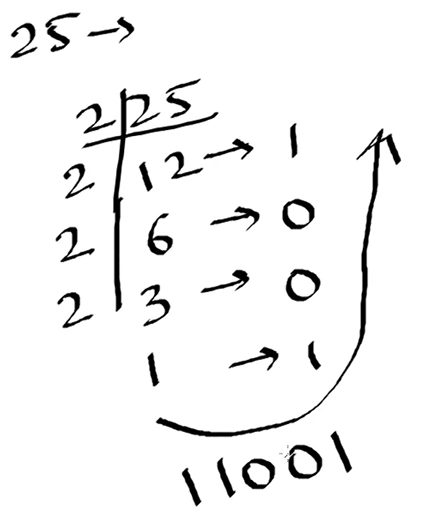
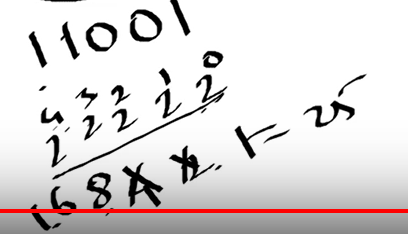
### Types

* Binary
* Decimal
* Octal
* HexaDecimal





### Convert Decimal to Binary and Binary to Decimal

## Import Module

import math

import math as m

from math import sqrt, pow

## User input in Python

Input function always get input in type str. We need to convert manually every time we get input.

x = int(input("Enter 1st number"))

y = int(input("Enter 2nd number"))

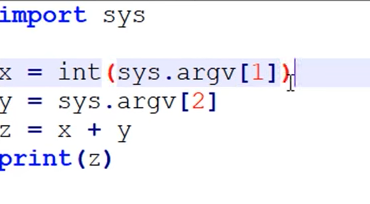
## Input from Command Line

We can use **agrv**

Index number of parameter when passing from command line



Index 0 is file name, and arguments starts at 1

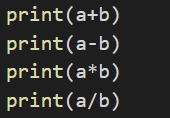
(we need to convert y too)



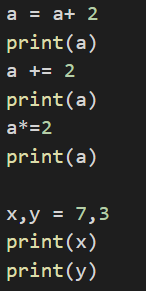
## Operators in Python

### Types

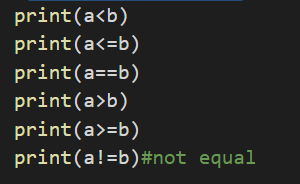
* Arithmetic Operator



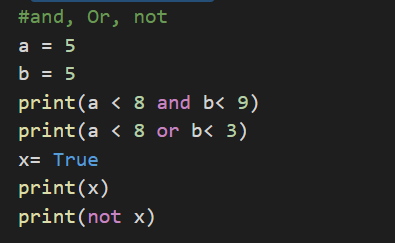
* Assignment Operator



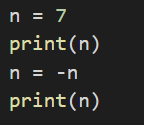
* Relational Operator



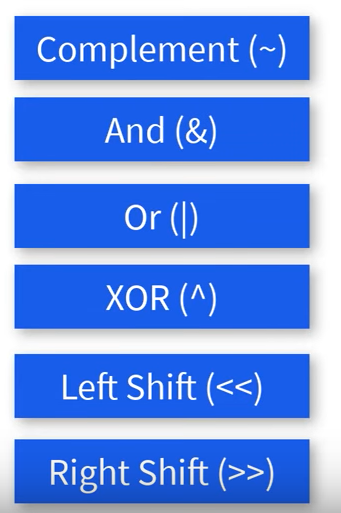
* Logical Operator



* Unary Operator



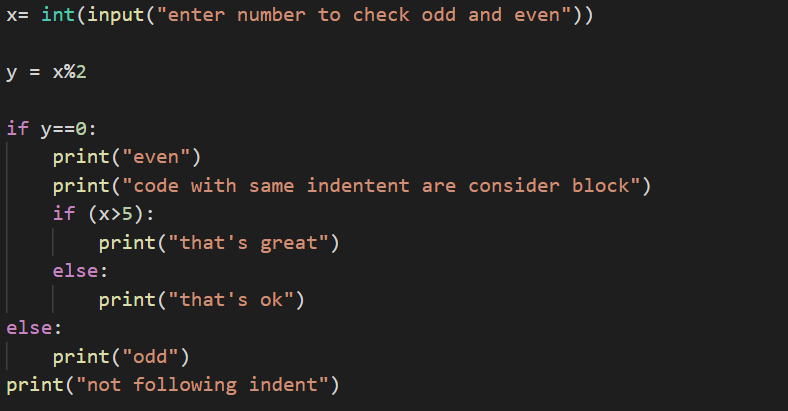
* Bitwise Operator



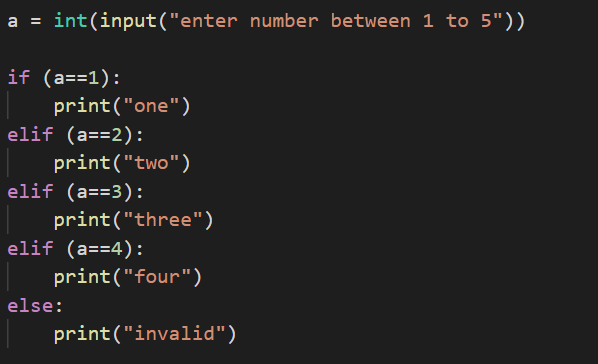
## Conditional Operation

If Elif Else Statement

Code block is identified by indent. All code with same indent as considered in same block.

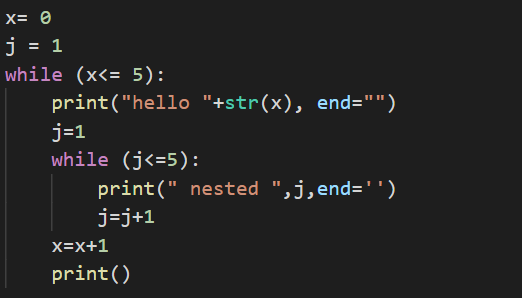


Elif

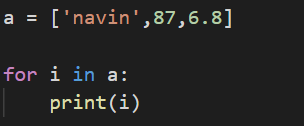


## Loops in Python

* While :
  + While loop has 3 parts variable with starting point, Condition and Incremental value.

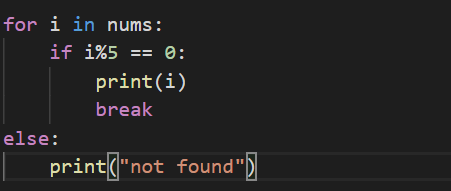


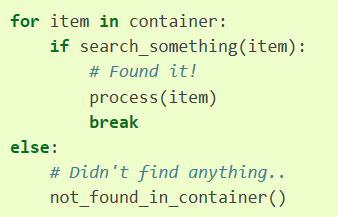
* For



* For else

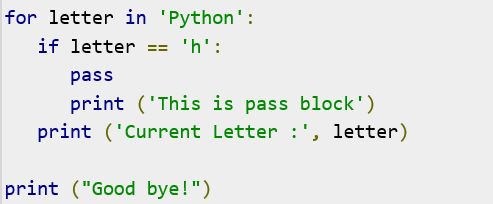
This is used when for loop execute without break condition is hit.





## Break Continue Pass in Python

* Break statement terminated the loops and comes out
* Continue will skip the execution below it and continue with next iteration of loop.
* Pass is used when a statement is required syntactically but you do not want any command or code to execute. We pass as same as empty brakets.

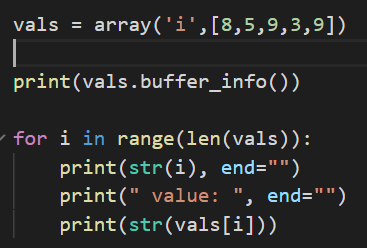
## Array in Python

* An array is a special variable, which can hold more than one value at a time.
* Type of objects stored in them is constrained. Array can hold only type of objects.
* Syntax

array.**array**(typecode[, initializer])

* This needs to be imported from package **array**

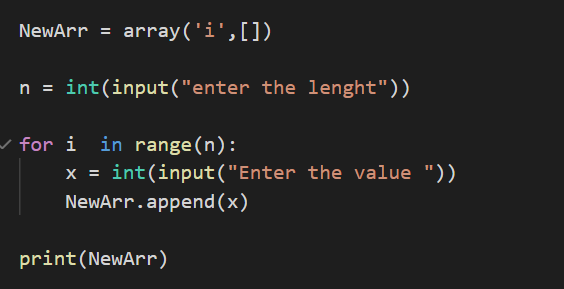




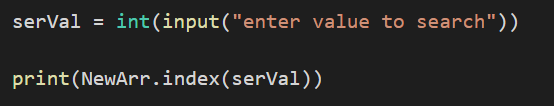
| **Type code** | **C Type** | **Python Type** | **Minimum size in bytes** |
| --- | --- | --- | --- |
| 'b' | signed char | int | 1 |
| 'B' | unsigned char | int | 1 |
| 'u' | Py\_UNICODE | Unicode character | 2 |
| 'h' | signed short | int | 2 |
| 'H' | unsigned short | int | 2 |
| 'i' | signed int | int | 2 |
| 'I' | unsigned int | int | 2 |
| 'l' | signed long | int | 4 |
| 'L' | unsigned long | int | 4 |
| 'q' | signed long long | int | 8 |
| 'Q' | unsigned long long | int | 8 |
| 'f' | float | float | 4 |
| 'd' | double | float | 8 |

### Inserting value into Array

To create empty array use empty [].



### Search value in Array



### Other Import functions in array

|  |  |
| --- | --- |
| array.**count**(x) | Return the number of occurrences of *x* in the array. |
| array.**extend**(iterable) | Append items from iterable to the end of the array. If iterable is another array, it must have exactly the same type code |
| array.**fromlist**(list) | Append items from the list. This is equivalent to **for x in list: a.append(x)** except that if there is a type error, the array is unchanged. |
| array.**index**(x) | Return the smallest *i* such that *i* is the index of the first occurrence of *x* in the array. |
| array.**insert**(i, x) | Insert a new item with value *x* in the array before position *i*. Negative values are treated as being relative to the end of the array. |
| array.**pop**([i]) | Removes the item with the index *i* from the array and returns it. The optional argument defaults to **-1**, so that by default the last item is removed and returned |
| array.**remove**(x) | Remove the first occurrence of *x* from the array. |
| array.**reverse**() | Reverse the order of the items in the array. |

## Multidimensional array.

We need to Numpy package for multidimensional array. Also u do not need datatype for array in numpy.

List = [ [1, 2], [2,5], [5,1] ]

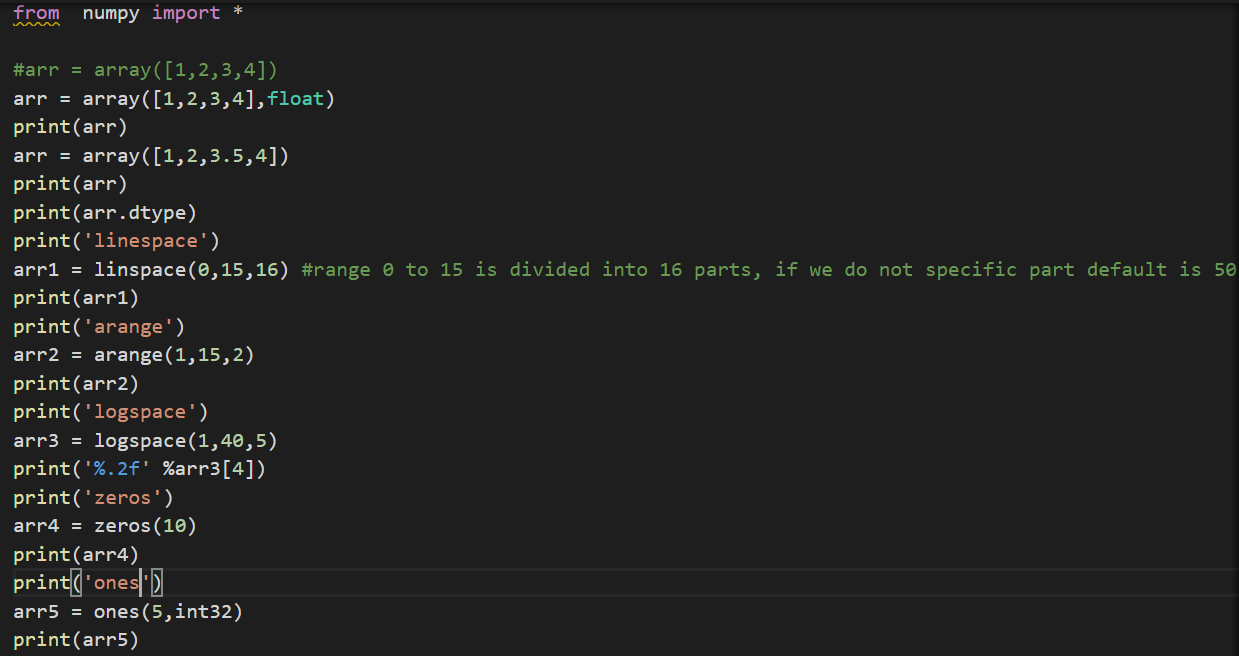
Print ( List[1] )

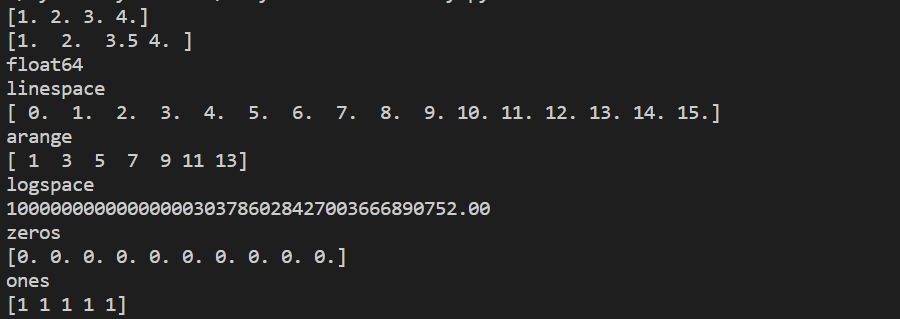
Output

[2, 5]

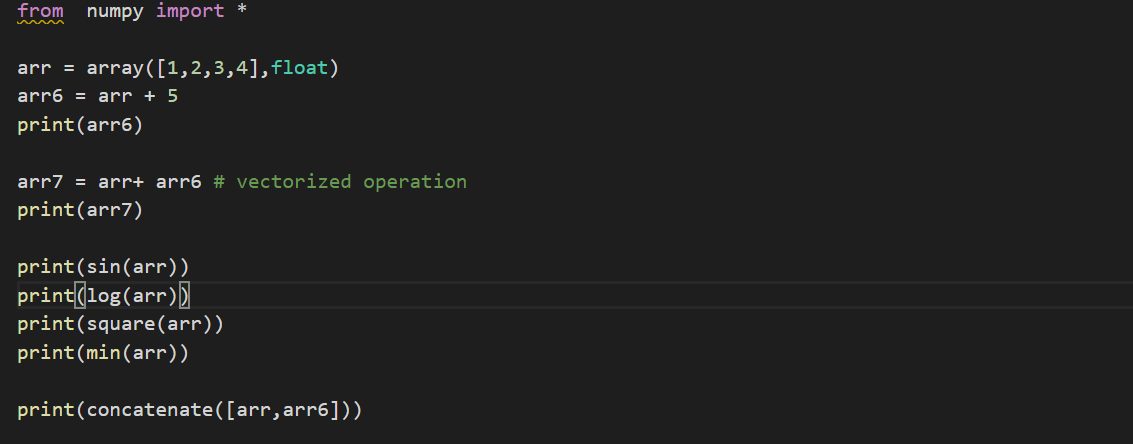
### Ways of Creating Arrays in Numpy

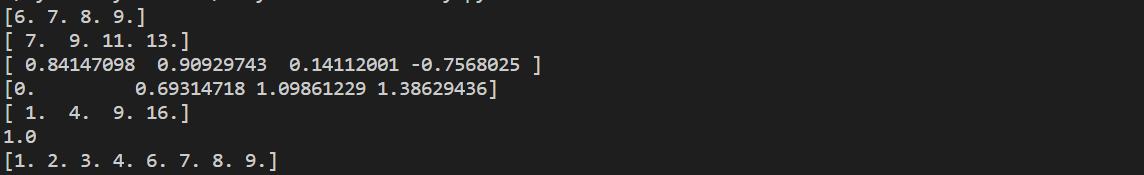
* Array()
* Linespace()
* Logspace()
* Arrange()
* Zeros()
* Ones()





### Operation in Array

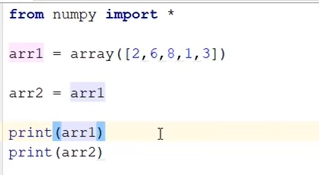
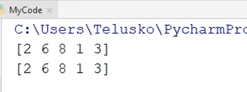




### Copying an Array

There are two type of copy

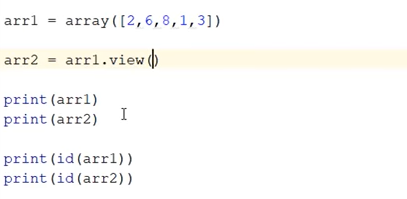
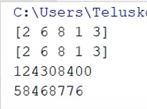
* Shallow copy
* Deep copy

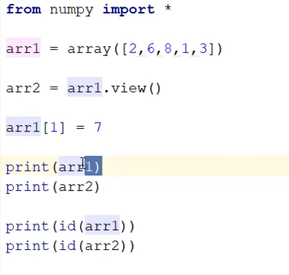
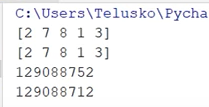
 

In this we will have array pointing to same address. So it’s a same array. Not really Copy.

#### Shallow Copy

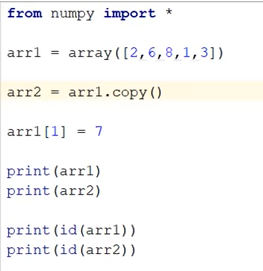
*Array.view() :-* This is shallow copy and both array are still dependent on each other. It create copy on different memory, but updating one array will also update another array.

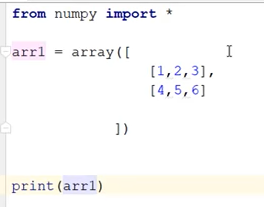
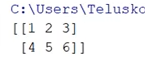
#### Deep copy

Array.copy()

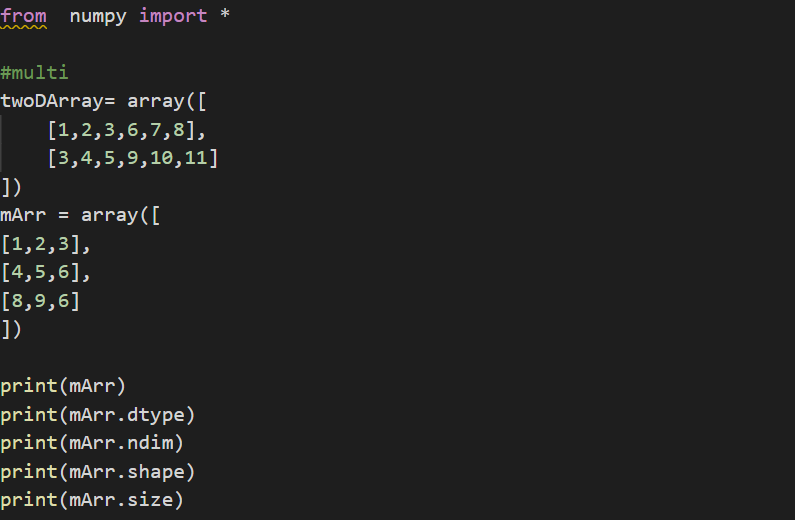
 

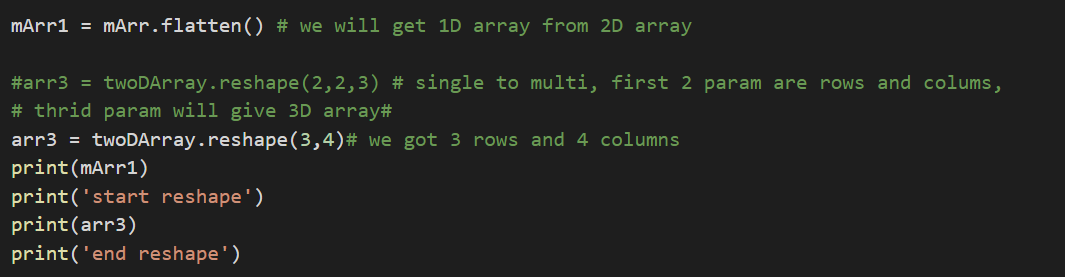
### Working with Matrix in Python

2D array

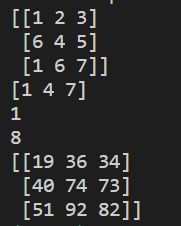
 

Few important function in MD array





Matrix Operation

# Function

## Create function

def greet():

    print("hello")

## Calling Function

greet()

## Argument & Return in Function



## Multiple return in function

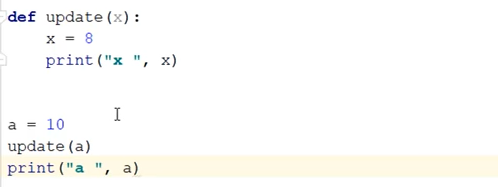


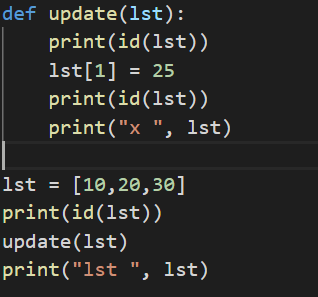
## Call by Reference and Call by Value

Pass by Value: When we pass value to function like eg here we are passing 10 to function not a, so x will get value as 10 and the update it to 8. Whereas a will still hold value of 10. This is pass by value.

Pass by Reference: we pass address of A to function. So when X is updated to 8, A will also update to 8.

Python does use both these method, instead value is called based in argument type, for example list is updated.

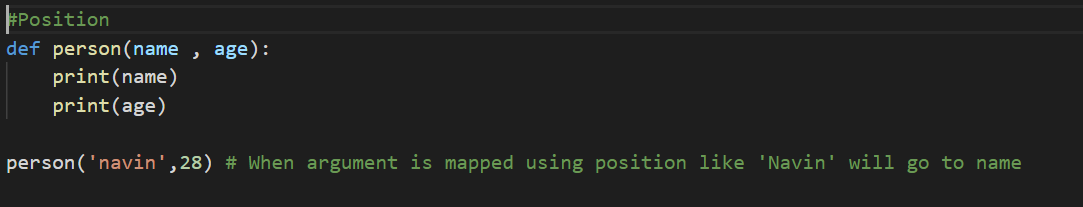
 

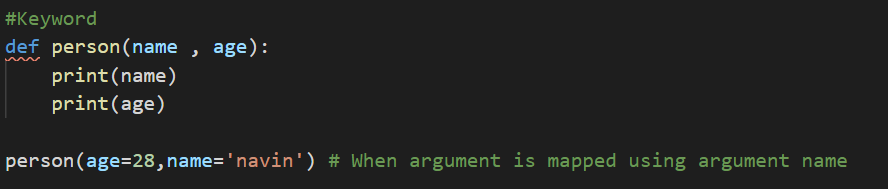
## Types of Arguments in Python

* Formal Arguments: defining argument at function creation.
* Actual Arguments: pass argument to function while calling function

### Position



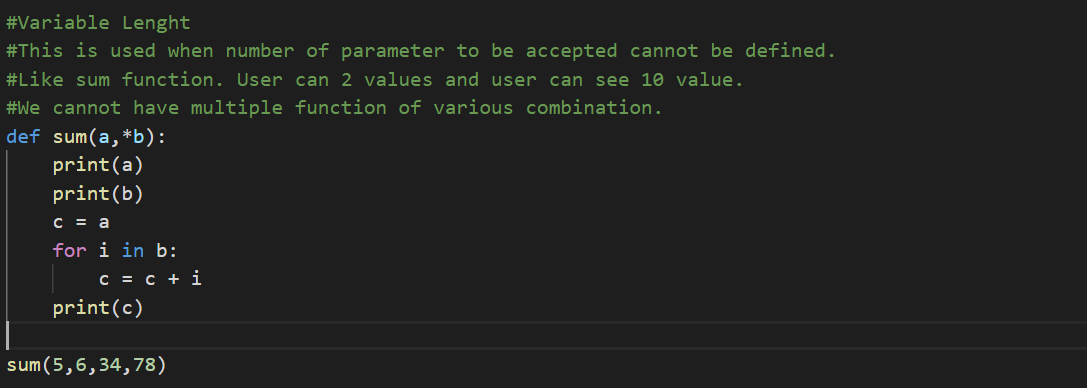
### Keyword



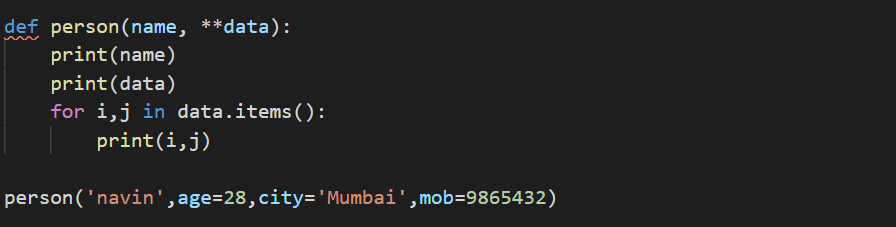
### Default

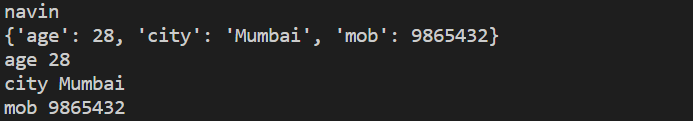


### Variable length

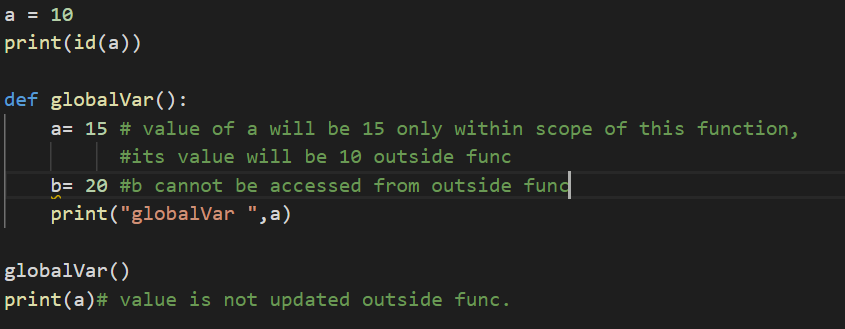


### Keyworded variable length argument in python

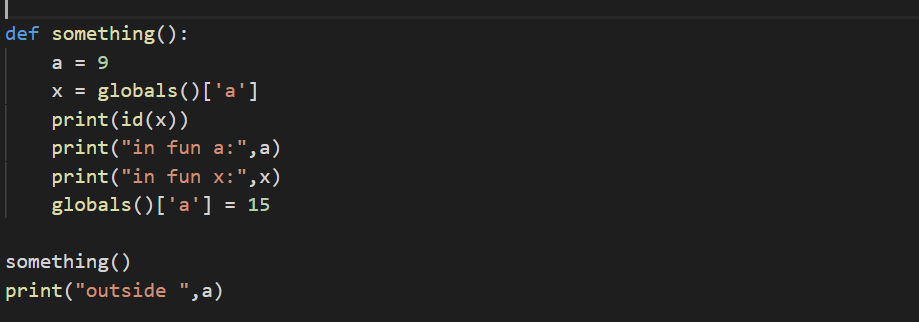


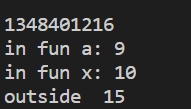


## Global Keyword in Python



To change value of global variable from function.





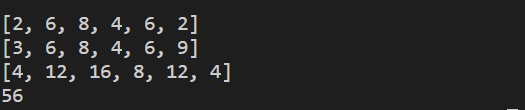
## Anonymous Functions

* We do not have name for these function.
* They are defined by Lambda



### Filter Map Reduce

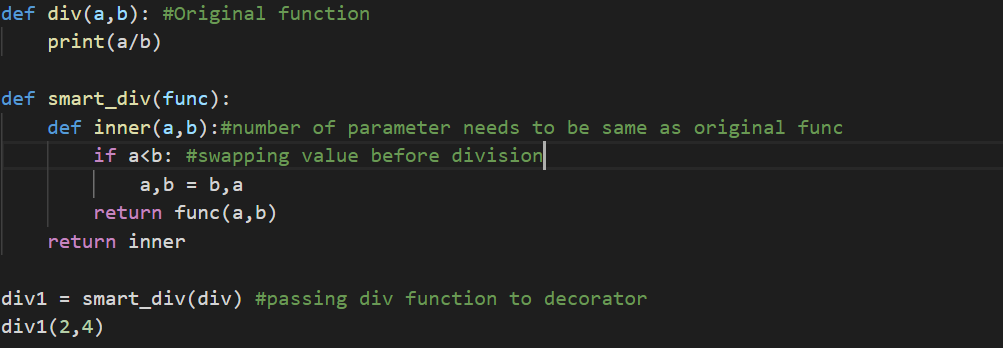




## Decorators

Using this we can add extra feature to existing function without making change to existing function.

Decorators allow you to inject or modify code in functions or classes

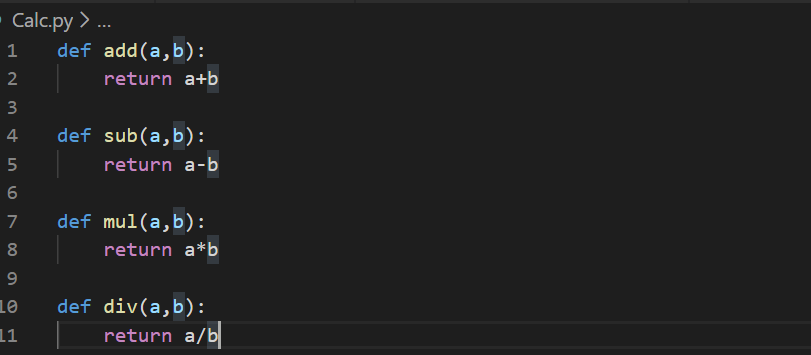


# Modules

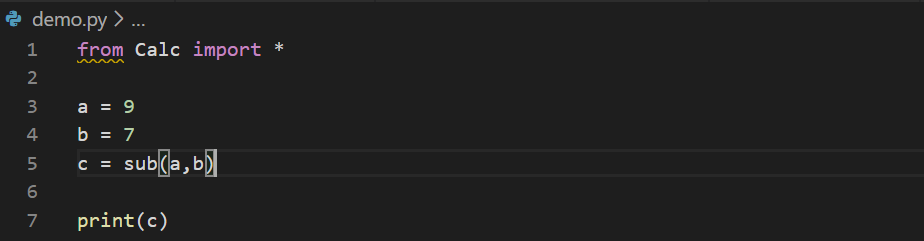
Diving code into separate logical group is modules. Like namespace in C#

## Creating Modules

Created a separate file for different functions of calculator

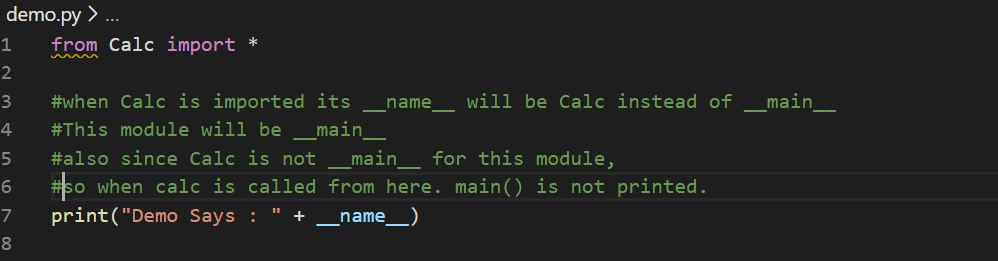


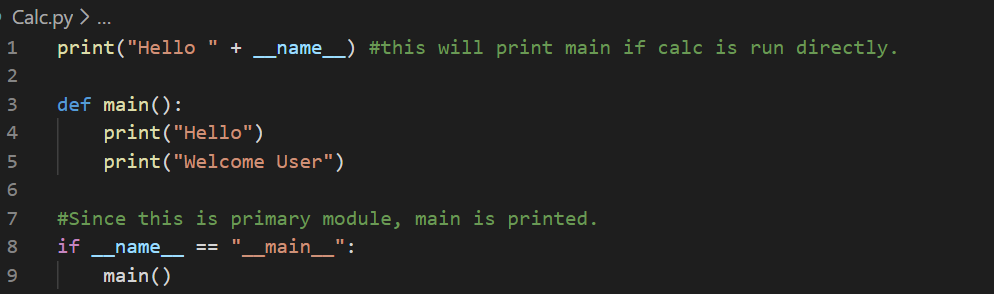
Calling Calc into file that needs to calculate value.



## Vairable\_\_name\_\_\_

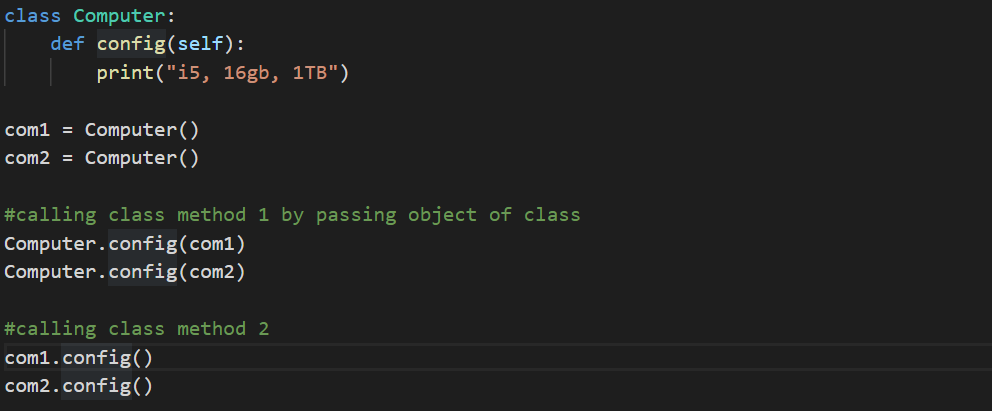
* The variable \_\_name\_\_ is inbuilt variable and hold value \_\_main\_\_ if that model is run directly. Else this variable will hold name of module if module is run from some other place.
* The first module name is always **\_\_main\_\_**. This is main function which is entry point of application.
* Any module in python can be Main. It is defined by module being used or called.
* Module which is importing other modules is considered main module.
* This is useful in identifying main module of application.





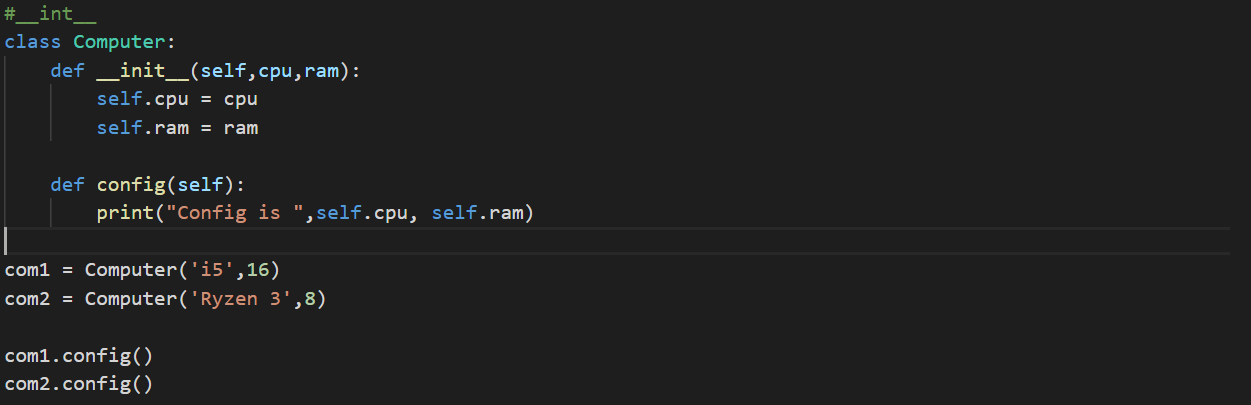
# Class

Class consist of variables and methods.



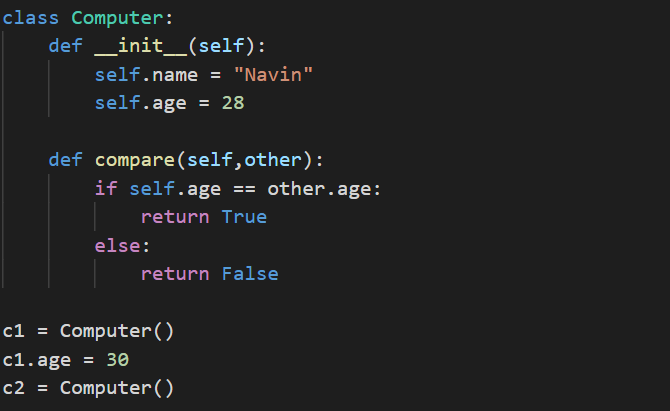
## \_\_int\_\_

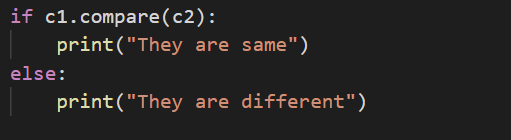
This is used for initializing variable for s class. Like parameterized constructor in C#. This is called automatically when class is called.



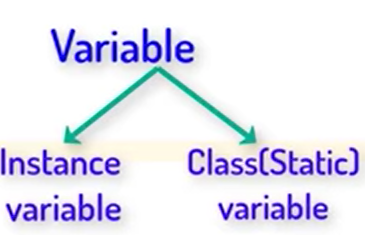
## Constructor, Self and Comparing Objects

* \_\_init\_\_ is constructor class. This called as soon as object of this class is called.
* \_\_init\_\_ defines variables to be used in class. Thus also define memory allocation of class.
* Self is used for defining objected being pass to class.
* For example. In below code compare method is called from objected of class c1. And as parameter object of same class is passed as c2 to compare. So self helps in identifying if method is called from c1 or c2

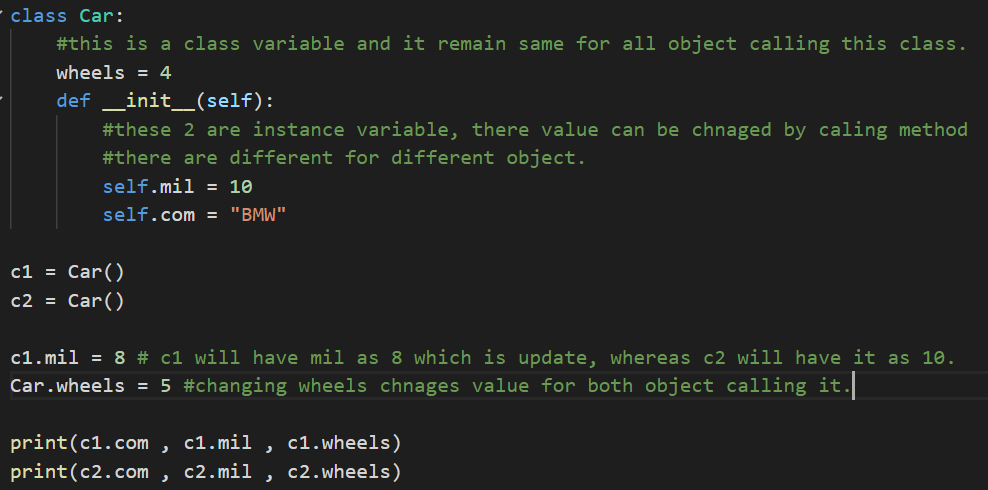




# Types of Variables



Instance variable are defined inside \_\_init\_\_. Class variable are defined outside \_\_init\_\_





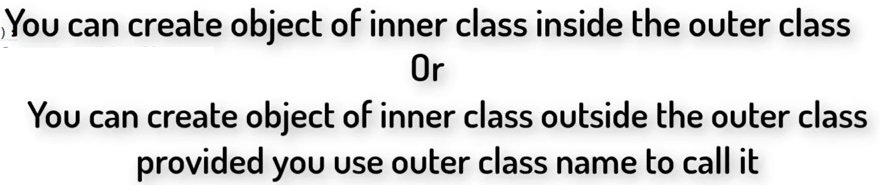
# Types of Methods

* Instance Method
  + Accessor Method : get method in C#
  + Mutator Method: set method in C#
* Class Method
* Static Method

|  |
| --- |
| class Student:      #this is a class variable, so it can only be used with class methods      school = 'Telusko'      def \_\_init\_\_(self,m1,m2,m3):          self.m1 = m1          self.m2 = m2          self.m3 = m3      # this is an instance method calls its calling and belong to object calling this method      def avg(self):          return (self.m1 + self.m2 + self.m3)/3      def get\_m1(self):# Accessor Method          return self.m1        def set\_m1(self,value): #Mutator Method          self.m1 = value      # this is a class methdo      # we need to use @classMethod symbol and cls parameter      @classmethod      def getSchool(cls):          return cls.school      @staticmethod      def info():          print("This is Student class.. in abc molude")  s1 = Student(34,47,32)  s2 = Student(89,32,12)  print(s1.avg())# callng instance method from an object.  #calling class method  print(Student.getSchool())  Student.school = "Huston"  print(Student.getSchool())  #calling static method  Student.info() |

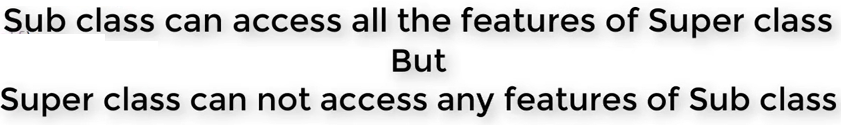
# Inner class

Inner class is used for providing logic grouping of class variables.



|  |
| --- |
| class Student:      def \_\_init\_\_(self,name,rollno):          self.name = name          self.rollno = rollno          self.lap = self.Laptop() # creating object of inner class on outer class      def show(self):          print(self.name , self.rollno)          self.lap.show()      #Inner class cannot be accessed directly from out side object.      #      class Laptop:          def \_\_init\_\_(self):              self.brand = "Hp"              self.cpu = 'i5'              self.ram = 8          def show(self):              print(self.brand,self.cpu,self.ram)  s1 = Student('Navin',2)  s2 = Student('Jenny',3)  s1.show()  lap1 = Student.Laptop()  print(s1.lap.brand) #access inner class variable |

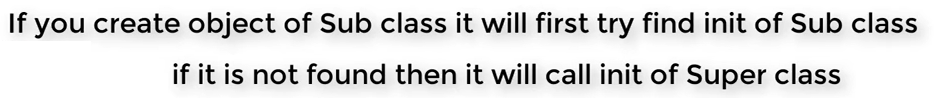
# Inheritance



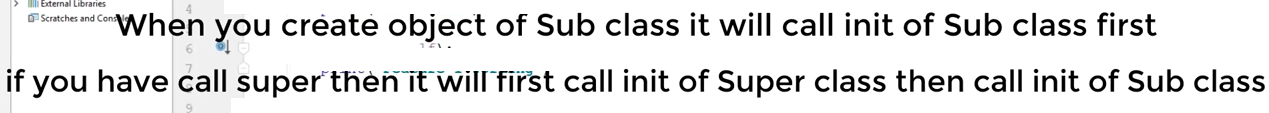
|  |
| --- |
| class A:      def feature1(self):          print("Feature 1 working")      def feature2(self):          print("Feature 2 working")  class B:      def feature3(self):          print("Feature 3 working")      def feature4(self):          print("Feature 4 working")  #Inheriting class A and B  class C(A,B):      def feature5(self):          print("Feature 5 working")  class D(A):      def feature6(self):          print("Feature 6 working")  class E(D):      def feature7(self):          print("Feature 7 working")  #A can access only feature of its own class  a1 = A()  a1.feature1()  a1.feature2()  #B can access only feature of its own class  b1 = B()  b1.feature3()  b1.feature4()  #C can access feature from all  A, B and C  c1 = C()  c1.feature3()  c1.feature1()  c1.feature5()  #D can access only feature from both A and D  d1 = D()  d1.feature1()  d1.feature6()  #E can access features from both A, D and E  e1 = E()  e1.feature1()  e1.feature6()  e1.feature7() |

## Constructor in Inheritance

* Constructor of super class is called when an object of sub class is initiated.
* For eg A is super class and B inherits A. When class B is called contracture of class A is also executed.



* For eg. If B is sub class and has \_\_Init\_\_ function then class B \_\_init\_\_ will be called instead of class A \_\_init\_\_



* If class C inherits class A first and the B. then calling Super method will call only \_\_init\_\_ of A instead of B. that is it will resolve from left to right. And class A is on left of class B.
* In order to call B we use **Method Resolution Order**.
* The same thing applies to methods. In case both A and B has method with same name. Then class A method will be given preference as class A is on left hand of class B.

|  |
| --- |
| class A:      def \_\_init\_\_(self):          print("in A Init")      def feature1(self):          print("Feature 1-A working")      def feature2(self):          print("Feature 2 working")  class B:      def \_\_init\_\_(self):          print("in B Init")      def feature3(self):          print("Feature 1-B working")      def feature4(self):          print("Feature 4 working")  class C(A,B):      def \_\_init\_\_(self):          super().\_\_init\_\_()          print("in C init")      #calling method of super class.      def feat(self):          super().feature2()  class D(A):      def \_\_init\_\_(self):          #this is used for calling super class \_\_init\_\_ function          super().\_\_init\_\_()          print("in D Init")      def feature6(self):          print("Feature 6 working")  a1 = C()  a1.feat()  d1 = D() |

# Polymorphism

## Uses

* Lose coupling
* Dependency Injection
* Interface

## Type

* Duck Typing
* Operator Overloading
* Method Overloading
* Method Overriding

## Duck Typing

* It only check if method which is being called is present in other class. It does not consider about logic or class that implements the method.
* For eg method execute is present in both class and being called by laptop class

|  |
| --- |
| class Pycharm:      def execute(self):          print("Compiling")          print("Running")  class MyEditor:      def execute(self):          print("Spell Check")          print("Convention Check")          print("Compiling")          print("Running")  class Laptop:      def code(self,ide):          ide.execute()  #class that being called here should have a method names execute  ide = MyEditor()  #ide = Pycharm()  lap1 = Laptop()  lap1.code(ide) |

## Operator Overloading

* When we use operators, a special function or magic function is automatically invoked that is associated with that particular operator.
* For example, when we use **+** operator, the magic method **\_\_add\_\_** is automatically invoked in which the operation for + operator is defined.
* So by changing this magic method’s code, we can give extra meaning to the + operator.

|  |
| --- |
| # a = '5'  # b = '6'  # print(a + b) #Magic method \_\_add\_\_ from str class is invoked.  # print(str.\_\_add\_\_(a,b))  class Student:      def \_\_init\_\_(self,m1,m2):          self.m1 = m1          self.m2 = m2      #We are adding input coming into \_\_add\_\_ method.      def \_\_add\_\_(self, other):          m1 = self.m1 + other.m1          m2 = self.m2 + other.m2          s3 = Student(m1,m2)          return  s3      def \_\_gt\_\_(self, other):          r1 = self.m1 + self.m2          r2 = other.m1 + other.m2          if r1 > r2:              return True          else:              return False      def \_\_str\_\_(self):          return '{} {}'.format( self.m1,self.m2)  s1 = Student(58,69)  s2 = Student(69,65)  #Here + operator will invoke \_\_add\_\_ method of Student class.  #output of s3 is 127 134,  s3 = s1 + s2  print(s3)  #Here > symbol will invoke \_\_gt\_\_ method  print(s3)  if s1 > s2:      print("s1 wins")  else:      print("s2 wins")  a = 9  print(a.\_\_str\_\_())  print(s2) |

List of Assignment operators and associated magic methods.

|  |  |
| --- | --- |
| **Assignment Operators** | **Magic Method or Special Function** |
| **-=** | **object.\_\_isub\_\_(self, other)** |
| **+=** | **object.\_\_iadd\_\_(self, other)** |
| **\*=** | **object.\_\_imul\_\_(self, other)** |
| **/=** | **object.\_\_idiv\_\_(self, other)** |
| **//=** | **object.\_\_ifloordiv\_\_(self, other)** |
| **%=** | **object.\_\_imod\_\_(self, other)** |
| **\*\*=** | **object.\_\_ipow\_\_(self, other)** |
| **>>=** | **object.\_\_irshift\_\_(self, other)** |
| **<<=** | **object.\_\_ilshift\_\_(self, other)** |
| **&=** | **object.\_\_iand\_\_(self, other)** |
| **|=** | **object.\_\_ior\_\_(self, other)** |
| **^=** | **object.\_\_ixor\_\_(self, other)** |

List of Comparison operators and associated magic methods.

|  |  |
| --- | --- |
| **Comparison Operators** | **Magic Method or Special Function** |
| **<** | **object.\_\_lt\_\_(self, other)** |
| **>** | **object.\_\_gt\_\_(self, other)** |
| **<=** | **object.\_\_le\_\_(self, other)** |
| **>=** | **object.\_\_ge\_\_(self, other)** |
| **==** | **object.\_\_eq\_\_(self, other)** |
| **!=** | **object.\_\_ne\_\_(self, other)** |

List of Binary operators and associated magic methods.

|  |  |
| --- | --- |
| **Binary Operators** | **Magic Method or Special Function** |
| **–** | **object.\_\_sub\_\_(self, other)** |
| **+** | **object.\_\_add\_\_(self, other)** |
| **\*** | **object.\_\_mul\_\_(self, other)** |
| **/** | **object.\_\_truediv\_\_(self, other)** |
| **//** | **object.\_\_floordiv\_\_(self, other)** |
| **%** | **object.\_\_mod\_\_(self, other)** |
| **\*\*** | **object.\_\_pow\_\_(self, other)** |
| **>>** | **object.\_\_rshift\_\_(self, other)** |
| **<<** | **object.\_\_lshift\_\_(self, other)** |
| **&** | **object.\_\_and\_\_(self, other)** |
| **|** | **object.\_\_or\_\_(self, other)** |
| **^** | **object.\_\_xor\_\_(self, other)** |

List of Unary operators and associated magic methods.

|  |  |
| --- | --- |
| **Unary Operators** | **Magic Method or Special Function** |
| **–** | **object.\_\_neg\_\_(self)** |
| **+** | **object.\_\_pos\_\_(self)** |
| **~** | **object.\_\_invert\_\_(self)** |

## Method Overloading

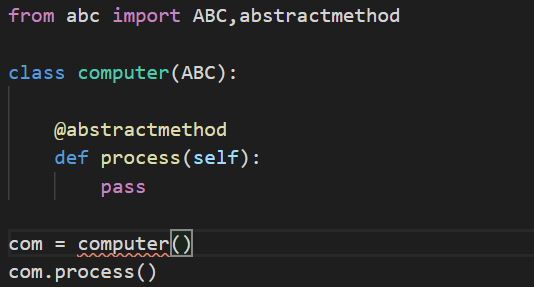
This in no concept of overloading in Python. Method with same name cannot be defined in class.

## Method Overriding

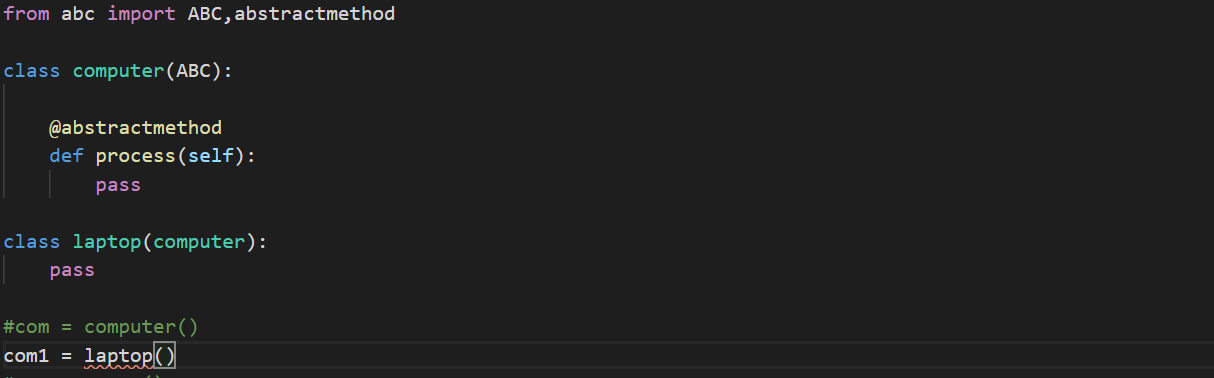
This is mainly useful while using inheritance.

# Abstract Class and Abstract Method

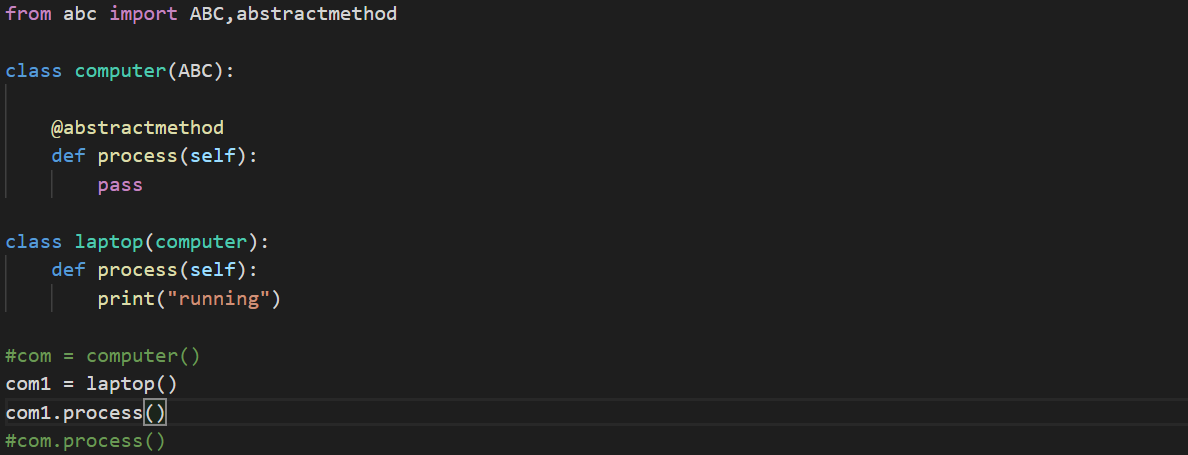
* Python does not as such support abstract class. We need a package needed ABC (Abstract base class)
* Abstract class has only definition and does not have implementation. And has at least one abstract method.
* Class inheriting Abstract class must implement all the abstract method.
* We cannot directly initiate abstract class.



* Subclass has to mandatorily implement abstract method else even sub class will be considered abstract class. The below code will throw and error.



* Below code will run perfectly



|  |
| --- |
| from abc import ABC,abstractmethod  class computer(ABC):      @abstractmethod      def process(self):          pass      def config(self):          print('min req: i5 16gb 1tb 1gb')  class laptop(computer):      def process(self):          print("running from laptop")  class desktop(computer):      def process(self):          print("running from desktop")  class programer:      def work(self,com):          print("solve bugs")          com.process()  #com = computer()  com1 = laptop()  com2 = desktop()  com1.config()  prog1 = programer()  prog1.work(com1)  prog2 = programer()  prog2.work(com2) |

# Iterator

* An iterator is an object that contains a countable number of values.
* An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.
* Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods \_\_iter\_\_() and \_\_next\_\_().
* Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable containers which you can get an iterator from.
* All these objects have a iter() method which is used to get an iterator.

|  |
| --- |
| mytuple = ("apple", "banana", "cherry")  myit = iter(mytuple)  print(myit.\_\_next\_\_())  print(next(myit))  print(next(myit)) |

* We can also use a for loop to iterate through an iterable object

|  |
| --- |
| mytuple = ("apple", "banana", "cherry")  myit = iter(mytuple)  for i in mytuple:      print(i) |

* Create own Iterator. To do so we need two important method iter() and next().
* For eg we want to get top 10 values.

|  |
| --- |
| class TopTen:      #counter variable num      def \_\_init\_\_(self):          self.num = 1      def \_\_iter\_\_(self):          return self      def \_\_next\_\_(self):          if self.num <= 10:              val = self.num              self.num += 1              return val          else:              raise StopIteration #raise the exception  values = TopTen()  print(next(values))  # print(values.\_\_next\_\_())  # print(values.\_\_next\_\_())  for i in values:      print(i) |

# Generators

* Similar to itieartor with iter() and next() method.
* Its uses keyword yield
* This is more memory efficient.

|  |
| --- |
| def topten():      yield 1      yield 2      yield 3      yield 4  values =  topten()  #we need to used next method to print yield  print(values.\_\_next\_\_())  print(values.\_\_next\_\_())  print('loop start')  #this loop will fetch value after first two print value done above  for i in values:      print(i)  print('loop end') |

* Get top ten perfect square root

|  |
| --- |
| def topten():      n = 1      while n <= 10:          sq = n\*n          yield sq          n += 1  values =  topten()  for i in values:      print(i) |

# Exception Handling

|  |  |
| --- | --- |
| try {     ......... } catch (<exceptionType> e) {     ........... } finally {     .......... } | try:   print(x)  except <exceptionType>:   print("x is not defined", e)  except Exception as e:   print("x is not defined", e) except:   print("Something went wrong") finally:   print("The 'try except' is finished") |
| a = 5  b = 2  try:      print("resource Open")      print(a/b)      k = int(input("Enter a number"))      print(k)  except ZeroDivisionError as e:      print("Hey, You cannot divide a Number by Zero" , e)  except ValueError as e:      print("Invalid Input")  except Exception as e:      print("Something went Wrong...")  finally:      print("resource Closed") | |

## Raise an exception

The raise keyword is used to raise an exception.

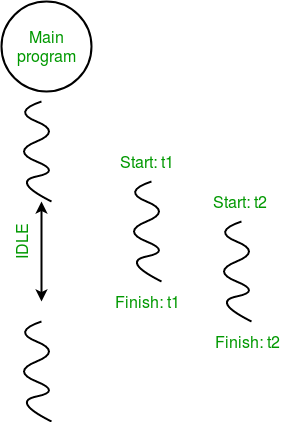
|  |
| --- |
| x = -1  if x < 0:   raise Exception("Sorry, no numbers below zero") |

You can define what kind of error to raise, and the text to print to the user.

|  |
| --- |
| x = "hello"  if not type(x) is int:   raise TypeError("Only integers are allowed") |

# MultiThreading

In python **Main Thread** is created along with addition threads. All class not in threads are executed by main threads.



|  |
| --- |
| from time import sleep  from threading import \*  class Hello(Thread):      def run(self):#run is inbuilt method for thread class; start() will call run() internally          for i in range(5):              print("Hello")              sleep(1)  class Hi(Thread):      def run(self):          for i in range(5):              print("Hi")              sleep(1)  t1 = Hello()  t2 = Hi()  t1.start()  sleep(0.2)  t2.start()  t1.join() #making main thread wait for t1  t2.join() #making main thread wait for t2  print("Bye") |

# File handling

file object = open(file\_name [, access\_mode][, buffering])

we import os for delete of file

|  |
| --- |
| import os  fo = open("FileHandling\_Foo.txt", "w")  fo.write( "Python is a great language.\nYeah its great!!\n")  fo.close()  fo = open("FileHandling\_Foo.txt", "a")  fo.write( "laptop")  fo.close()  f = open('FileHandling\_Foo.txt','r')  print(f.read())  print(f.readline())  print(f.readline(4),end="")  f.close()  f = open('FileHandling\_Foo.txt','rb')  f1 = open('FileHandling\_Foo1.txt','wb')  for i in f:      f1.write(i)  f.close()  f1.close()  if os.path.exists("FileHandling\_Foo.txt"):    os.remove("FileHandling\_Foo.txt")  else:    print("The file does not exist") |

# Json

|  |
| --- |
| import json  # a Python object (dict):  x = {    "name": "John",    "age": 30,    "city": "New York"  }  # convert into JSON:  y = json.dumps(x)  # the result is a JSON string:  print(y) |

# XML

# Date

Import date

|  |
| --- |
| import datetime  x = datetime.datetime.now()  print(x.year)  print(x.strftime("%A"))  x = datetime.datetime(2020, 5, 17)  print(x)  x = datetime.datetime(2018, 6, 1)  print(x.strftime("%B")) |

# RegEx

Import re

|  |
| --- |
| import re  txt = "The rain in Spain"  x = re.search("^The.\*Spain$", txt)  print(x)  txt = "The rain in Spain"  x = re.findall("ai", txt)  print(x)  txt = "The rain in Spain"  x = re.search("\s", txt)  print("The first white-space character is located in position:", x.start())  txt = "The rain in Spain"  x = re.split("\s", txt)  print(x) |

# Zip

The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.

If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.

|  |
| --- |
| a = ("John", "Charles", "Mike")  b = ("Jenny", "Christy", "Monica", "Vicky")  x = list(zip(a, b))  y= set(zip(a,b))  z = dict(zip(a,b))  print(x)  print(y)  print(z)  for a,b in x:      print(a, b) |

# Connect to SQL Server Database

Step 1: Install pyodbc

Pip install pyodbc

Step 2: Retrieve the server Details.

Step 3: Connect Python to SQL Server

|  |
| --- |
| import pyodbc  conn = pyodbc.connect('Driver={SQL Server};'  'Server=server\_name;'  'Database=db\_name;'  'Trusted\_Connection=yes;')  cursor = conn.cursor()  cursor.execute('SELECT \* FROM db\_name.Table')  for row in cursor:  print(row) |

# Socket Programming

We need to include package named **socket**.

## Port Number

Define internal address on same server. For eg a server can host multiple services. Each services will have their own port number.

## Type of network

* TCP (Transmission Control Protocol): connection oriented, means we need to first create connection and then communicate.
* UDP (User Datagram Protocol): connection less, in this we do not need to establish connection we just send the package to respected ip address. Disadvantage is we are not sure if packet has reached.

Server Code

|  |
| --- |
| import socket   # we need to pass to variable IP type and connetion type,   # default IP type is IPv4 and connection type is tcp  s = socket.socket()  print('Socket Created')  #define address and port number of  s.bind(('localhost',9955))  #listen to max 3 connection and refuse any further connection  s.listen(3)  print('waiting for connection')  #keep connection open  while True:      c,addr= s.accept()      #recevice data from client      name = c.recv(1024).decode()      print('connected with: ', addr, name)        #send data to client      c.send(bytes('Welcome','utf-8'))      c.close( |

Client Side Programing

|  |
| --- |
| import socket  c= socket.socket()  #establish connection with server  c.connect(('localhost',9955))  #send data to server  name = input("Enter your name")  c.send(bytes(name,'utf-8'))  #get data from server  print(c.recv(1024).decode()) |

GitHub

Documentation: <https://code.visualstudio.com/docs/editor/versioncontrol#_remotes>

# Add an existing vs code project to git and github

Step 1: Install git on your system

<https://git-scm.com/downloads>

Step1.1: Configure GitHub.

Open Github cmd and type below commands

Git config –global user.email “<email Address>”

Git config –global user.name “<githug name>

Step 2: Login to GitHub

Step 3: Create a repository on GitHub & copy URL

Step 4: Goto VS Code and open project/folder

Note: check git is enabled from settings

Step 5: Goto source control section & click on git icon to initialize github.

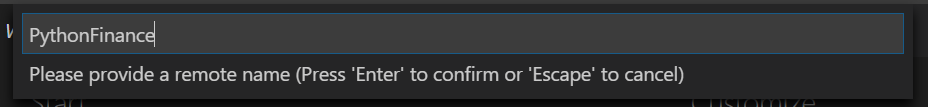
Step 6: Give commit message & Commit the changes

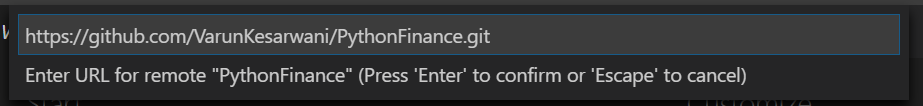
Step 6.1: setup git on vs code. Open terminal and type command

git remote add origin https://github.com/VarunKesarwani/PythonFinance.git

Step 7: Add remote repo (github repo) on VS Code (ctrl+shift+P)







Step 8: Push committed changes to github repo

Note: It might ask you username and password on first push

Step 9: Check changes on github repo

# Merge project from github

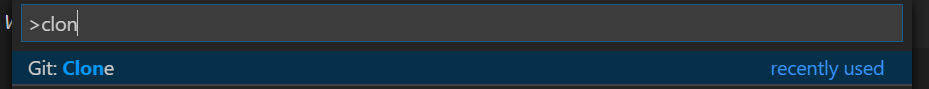
Use below command to merge project with existing git project. Make sure project structure for merging should be same.

* git pull origin master --allow-unrelated-histories
* git merge origin origin/master
* git pull origin master

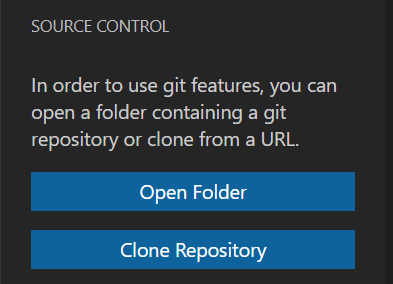
# Clone from github to vscode

Step 1: Copy URL from Github repo.

Step 2: Open Command Pallete.



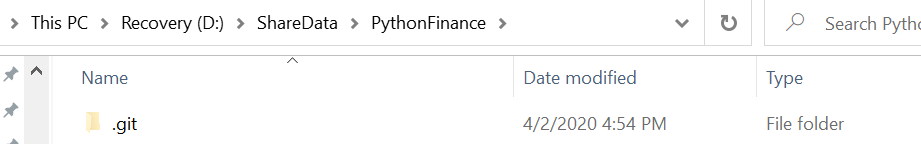
(or)



Step 3: Select Local folder for downloading file. This will be treated as git repo.

# Remove project from git

Delete .git hiden folder from project path.



Anaconda

* Anaconda is a distribution of Python
* A distribution contains various python libraries and make sure they work well together.
* Some of the more complex libraries have certain dependencies and Anaconda makes sure these are working.

# Jupyter Notebook

Jupyter notebook have individual cell to execute python code

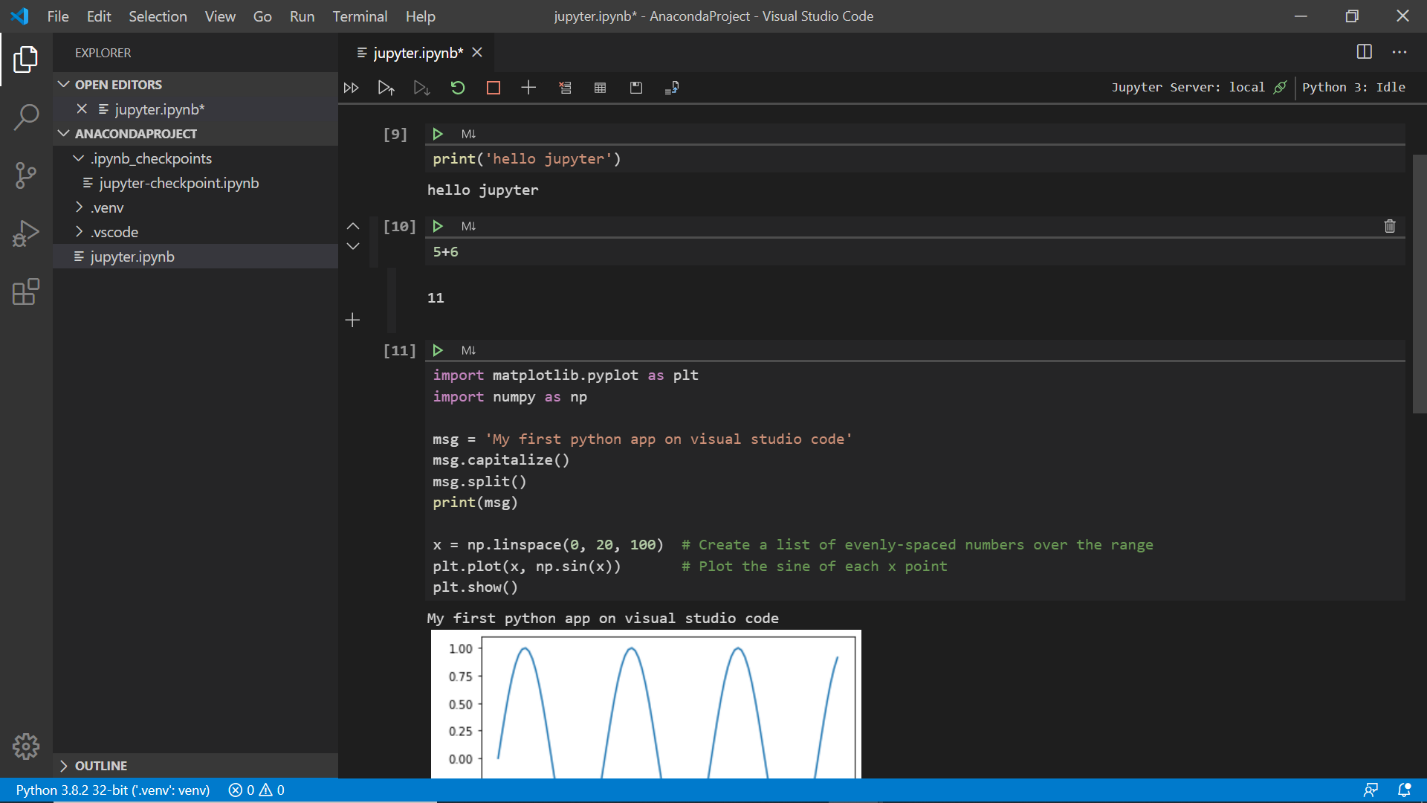
## On Anaconda

* Open Anaconda Navigator. And click on **JupyterLab** launch
* Open Anaconda Cmd type cmd: jupyter notebook
* Both these steps will launch browser. And all files will show up on browser.

## On Code

Install jupyter using code: pip install jupyter

If you open a Jupyter notebook file (.ipynb) in VS Code, you can use the Jupyter Notebook Editor to directly view, modify, and run code cells.



## Setting Jupyter environment variable

There should be a file environment.yml on directory. Then type command on anaconda prompt. This will install required package for project

Conda env create –f environment.yml

Django

# Django is a high-level Python-based free and open-source web framework, which follows the **model-view-template (MVT)** architectural pattern.

Django's primary goal is to ease the creation of complex, database-driven websites

# Setup

* Create project and virtual environment like above. And open project in Code.
* Run follow command in terminal
  + *pip install Django*
* Check verison
  + *django-admin –version*
* Start a project
  + *django-admin startproject PythonWebPage*
  + navigate to project folder “*cd PythonWebPage*”
  + There are multiple files being generated by python once we create project.
  + And one those file is manage.py
* Manage.py is main page and u can run the server. Use below command to run server
  + *Python manage.py runserver*

Rasa

# Installation

* Install Python version 3.7 or less. Uninstall all the other versions.
* Set variable path
  + setx path “%PATH%; C:\Users\vgupta\AppData\Local\Programs\Python\Python37\Scripts”
* Install pip using windows command
  + Navigate to python setup path in CMD and run below command
  + python -m pip install --upgrade tensorflow
* Install Visual C++ 14 using visual studio installer.
* Create project and virtual environment as mentioned above.
* Run command
  + Pip install rasa
* Install Pipe line.
  + pip install rasa[spacy]
* Create Rasa project
  + rasa init --no-prompt
  + this will create various files required for rasa project
* view your nlu training data
  + cat data/nlu.md
* Define your model configuration
  + cat config.yml
* Write your first stories
  + cat data/stories.md
* Define a Domain
  + cat domain.yml

So what do the different parts mean?

|  |  |
| --- | --- |
| intents | things you expect users to say |
| actions | things your assistant can do and say |
| templates | template strings for the things your assistant can say |

* Train a Model
  + rasa train
  + echo "Finished training."
* Test ur Assistant
  + rasa test
* Talk to ur Assistant
  + rasa shell

Python for Finance

# Required libraries

* Numpy: *pip install numpy*
* Pandas : *python -m pip install pandas*
* Pandas-datareader: *python -m pip install pandas-datareader*
* Matplotlib : *python -m pip install matplotlib*
* Beautifulsoup4 : *pip install beautifulsoup4*
* Sklearn: *pip install sklearn*

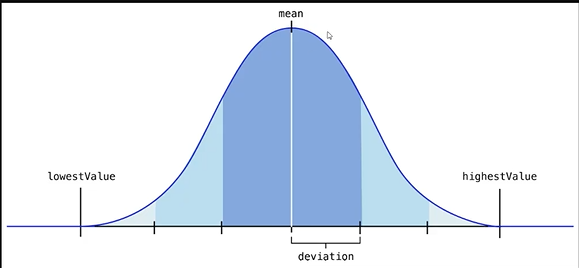
Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

* a powerful N-dimensional array object
* matrix operations
* sophisticated (broadcasting) functions
* tools for integrating C/C++ and Fortran code
* Useful linear algebra, Fourier transform, and random number capabilities.
* Arrays are efficient the list
* Run command Pip install Numpy

# Types

* Arrays (single and Multi)
* Arange
* Zeros
* Ones
* Linespace
* Eye
* Random
  + Uniform distribution
  + Standard distribution



# Array Function

* Reshape
* Shape
* Dtype (data type)
* Max
* Argmax
* Min
* Argmin

# Operation in Array

|  |
| --- |
| import numpy as np  arr = np.arange(0,10)  print(arr)  print(arr+arr)  print(arr\*arr)  print(arr - arr)  print(arr / arr)#shows warning for divide by zero  print(1/arr)  print(arr\*\* 3)  print(arr+ 10)  print(np.sqrt(arr))  print(np.exp(arr))  print(np.max(arr))  print(np.sin(arr))  print(np.log(arr)) |

# Numpy Indexing

|  |
| --- |
| import numpy as np  arr = np.arange(0,11)  print(arr)  print(arr[2])  print(arr[2:5]) #Output: [2 3 4]  print(arr[:4]) #Output: [0 1 2 3]  print(arr[5:]) #Output: [ 5  6  7  8  9 10]  #boardcast  arr[0:4] = 50  print(arr) #Output : [50 50 50 50  4  5  6  7  8  9 10]  arr2 = np.arange(0,11)  slice\_of\_arr = arr2[0:6]  print(slice\_of\_arr)  slice\_of\_arr[:] = 99  print(slice\_of\_arr)  print(arr2)  arr\_copy = arr2.copy()  arr\_copy[:] = 200  print(arr\_copy)  print(arr2)  #matrix index  #mat[row,col]  #mat[row][col]  mat = np.array([[23,54,76],[52,12,65],[67,34,48]])  print(mat)  print(mat[0])#get all elememt in row 0  print(mat[2])#get all elememt in row 2  print(mat[2][2])#get all elememt in row 0 and col 2  print(mat[2,2])#get all elememt in row 0 and col 2  #Slice mat  mat\_slice = mat[:2,1:]  print(mat\_slice)  mat\_slice2 = mat[1:,:2]  print(mat\_slice2)  #conditional selection  arr3 = np.arange(0,11)  print(arr3 > 4)  #output [False False False False False  True  True  True  True  True  True]  bool\_arr = arr3 > 4  print(arr3[bool\_arr])# return all value when index is true  #output [ 5  6  7  8  9 10]  print(arr3[arr3>4])  #output [ 5  6  7  8  9 10] |

Pandas

Pandas was created with finance in mind.

* Dataframe object
* Reading and writing data between in-memory data structure and different format like csv, excel, sql and fast HDF5 format, etc.
* Visualization libraries.
* High optimized
* Install: pip install pandas

# Series

Similar to Numpy array, except we can give them a named or datetime index, instead of just numerical index

We pass two argument **DATA** and **INDEX**

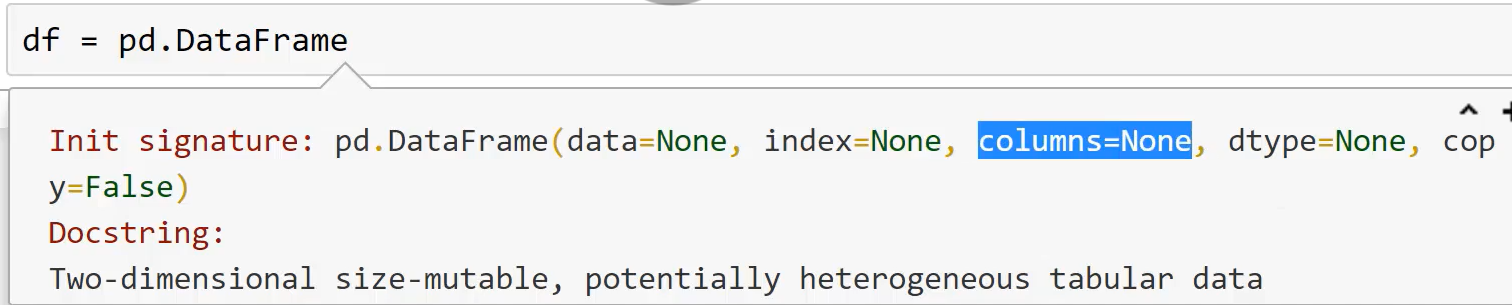
|  |
| --- |
| labels = ['a','b','c']  print(pd.Series(data=my\_list))  my\_list = [10,20,30]  #with Index  print(pd.Series(data=my\_list,index=labels)) |

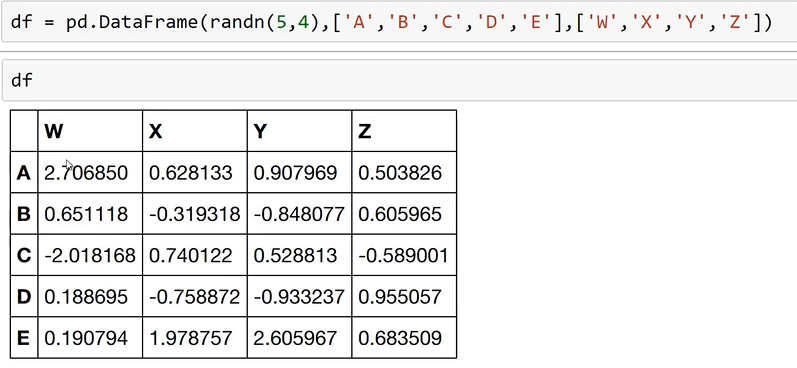
# DataFrames

DataFrames are the workhorse of pandas and are directly inspired by the R programming language. We can think of a DataFrame as a bunch of Series objects put together to share the same index. Let's use pandas to explore this topic

Syntax

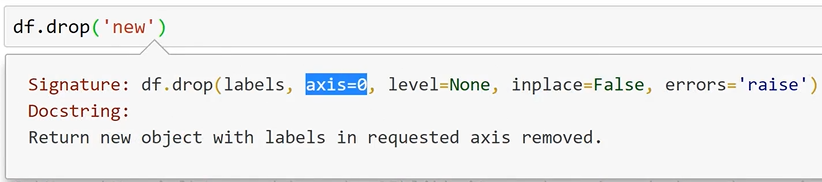
*pandas.DataFrame(<data>, <index/rows>, <Columns>, <dtype>)*





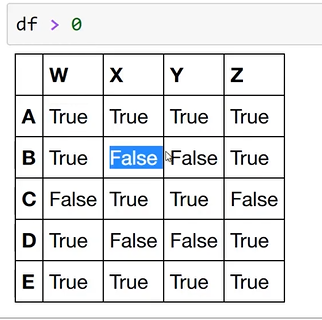
Dataframes generates data in rows and column format.

Axis: denotes 0 as row and 1 as column. This is because dataframe is index markup of array. For eg below created dataframe shape will return Tuple of 5, 4; where 5 is no of rows and 4 is No of Col. And thus axis index 0 is row and 1 is columns.



|  |
| --- |
| #Create Df  df = pd.DataFrame(randn(5,4),['A','B','C','D','E'],['W','X','Y','Z'])  print(df) |
| #Get Column from DF  print(df['W'])#This is Standard  print(df.W)  print(type(df['W'])) #<class 'pandas.core.series.Series'>  print(type(df)) #<class 'pandas.core.frame.DataFrame'>  #Get multiple Column from DF  print(df[['W','Z']])# Pass list of column  #Create new column and column operation  df['New'] = df['W']+ df['Y']  print(df) |
| #Drop columns and Axis  #this will not actually drop table. This will only drop on display  df.drop('New',axis = 1)  print(df)  # this will actually drop on table with parameter inplace = True  df.drop('New',axis = 1, inplace = True)  print(df)  #Drop rows  print(df.drop('E',axis = 0))#With out inplace  print(df) |
| # Selecting Rows  print(df.loc['C'])  print(df.iloc[2]) |
| #Subset of table  print(df['W']['A'])  print(df.loc['C','Y'])  print(df.loc[['A','B'],['W','Y']]) |

# Conditional Selection

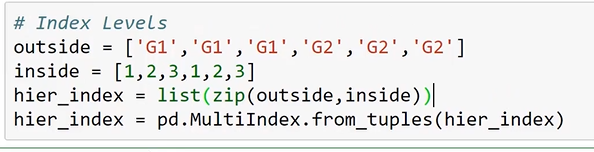


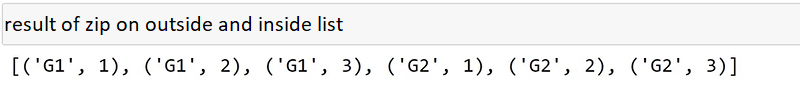
|  |
| --- |
| boolDf = df > 0  print(boolDf)  print(df[boolDf])#NaN is false  print(df[df>0])  #Subset conditional selection  print(df[df['W']> 0]) # will not get row C, meaning. df only get series value with True  print(df[df['Z']< 0])  res = df[df['W']> 0]  print(res['X'])  print(df[df['W']> 0]['Y']) #single steps  print(df[df['W']> 0][['Y','X']]) #single steps  #Multiple Condition  #use & instead of 'and'  print(df[(df['W']> 0) & (df['Y']> 1)]) #And  print(df[(df['W']> 0) | (df['Y']> 1)]) #OR |

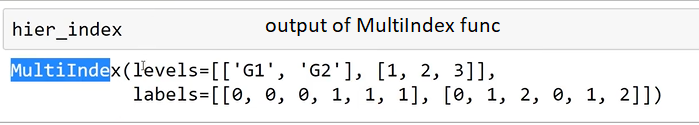
# Set and Reset Index

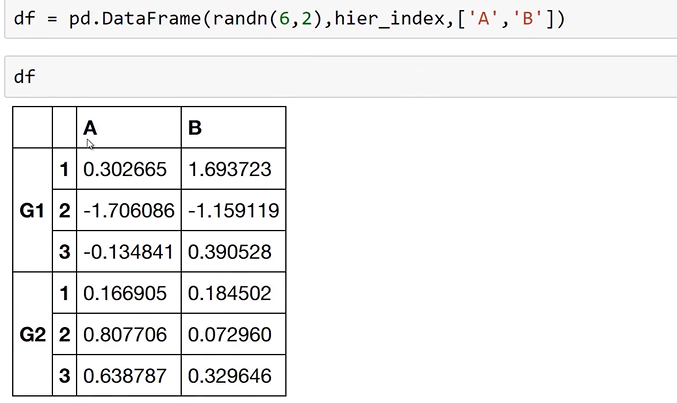
|  |
| --- |
| print(df)  print(df.reset\_index())#inplace = true for hard commit  newInd = 'UP TN AP HP RJ'.split() #creating list by splitting with ' '  print(newInd)  df['State'] = newInd #adding new col to df  print(df)  df.set\_index('State') #This will update original index A,B,C,D,E; inplace = true for hard commit  print(df) |

# Multi Hierarchy Index Dataframe

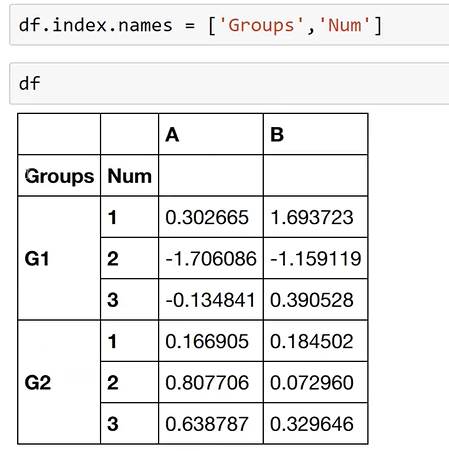




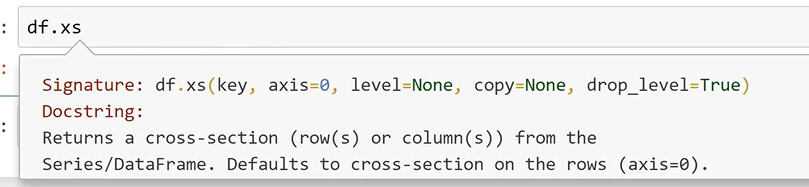


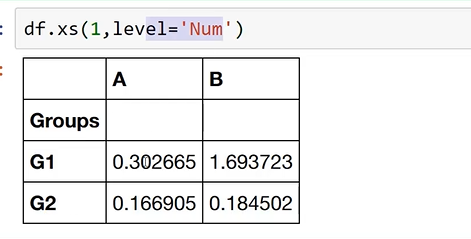


# Index Name



# Cross Section

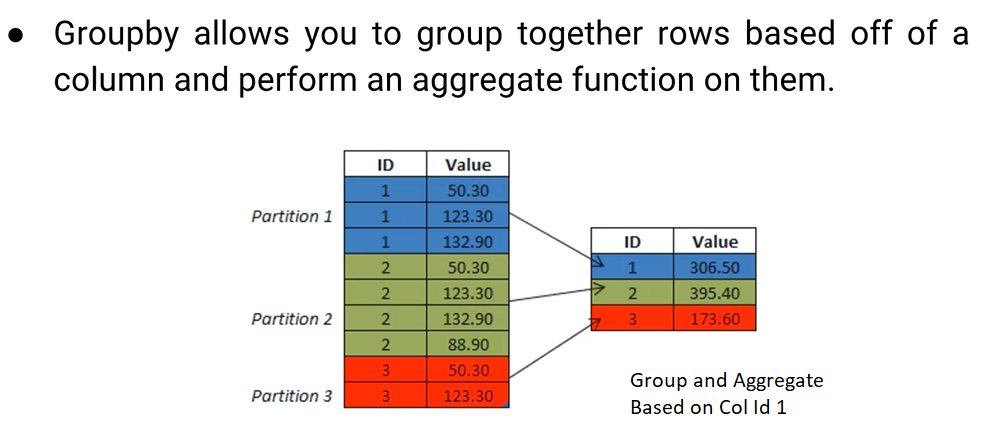




# Missing Data

|  |
| --- |
| import numpy as np  import pandas as pd  d = {'A':[1,2,np.nan],'B':[5,np.nan,np.nan],'C':[6,7,8]}  df= pd.DataFrame(d)  print(df)  #dropna  #dataframe will drop rows if it has null value.  #row 1 and 2 wil be dropped as they have Nan value; axis parameter is 0 which is default  print(df.dropna())  #To perform this on column, specific axis parameter to 1  print(df.dropna(axis=1))  #we can also set treshold value when ro drop row or col.  #for example we want to drop only when row or col has more the 2 nan value  print(df.dropna(axis=0,thresh=2))  #Fill Nan  print(df.fillna(value='Fill'))  #mean of value  print(df.fillna(value=df['A'].mean())) |

# Group By



|  |
| --- |
| import numpy as np  import pandas as pd  data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],         'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],         'Sales':[200,120,340,124,243,350]}  df = pd.DataFrame(data)  #GroupBy Column Company  by\_comp = df.groupby('Company')  #these will only consider column with numeric value  print(by\_comp.mean())  print(by\_comp.sum())  print(by\_comp.std())  print(by\_comp.sum().loc['FB'])  by\_comp1 = df.groupby('Company').sum().loc['FB']  print(by\_comp1)  print(by\_comp.count())  print(by\_comp.max())  print(by\_comp.min())  print(by\_comp.describe())  print(by\_comp.describe().transpose()) |

# Merging, Joining and Concatenating

## Concatenation

Concatenation basically glues together DataFrames. Keep in mind that dimensions should match along the axis you are concatenating on. You can use pd.concat and pass in a list of DataFrames to concatenate together:

|  |
| --- |
| import pandas as pd  df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],                          'B': ['B0', 'B1', 'B2', 'B3'],                          'C': ['C0', 'C1', 'C2', 'C3'],                          'D': ['D0', 'D1', 'D2', 'D3']},                          index=[0, 1, 2, 3])  df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],                          'B': ['B4', 'B5', 'B6', 'B7'],                          'C': ['C4', 'C5', 'C6', 'C7'],                          'D': ['D4', 'D5', 'D6', 'D7']},                           index=[4, 5, 6, 7])  df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],                          'B': ['B8', 'B9', 'B10', 'B11'],                          'C': ['C8', 'C9', 'C10', 'C11'],                          'D': ['D8', 'D9', 'D10', 'D11']},                          index=[8, 9, 10, 11])  print(pd.concat([df1,df2,df3])) #on Row  print(pd.concat([df1,df2,df3],axis=1)) #on Col |

## Merging

The merge function allows you to merge DataFrames together using a similar logic as merging SQL Tables together. For example:

|  |
| --- |
| import pandas as pd  left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],                       'A': ['A0', 'A1', 'A2', 'A3'],                       'B': ['B0', 'B1', 'B2', 'B3']})    right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],                            'C': ['C0', 'C1', 'C2', 'C3'],                            'D': ['D0', 'D1', 'D2', 'D3']})  print('Merge')  #We are merging on key col  #inner is default for merge  print(left)  print(right)  print(pd.merge(left,right,how='inner',on='key'))  left1 = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],                       'key2': ['K0', 'K1', 'K0', 'K1'],                          'A': ['A0', 'A1', 'A2', 'A3'],                          'B': ['B0', 'B1', 'B2', 'B3']})    right1 = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],                                 'key2': ['K0', 'K0', 'K0', 'K0'],                                    'C': ['C0', 'C1', 'C2', 'C3'],                                    'D': ['D0', 'D1', 'D2', 'D3']})  print(pd.merge(left1,right1,on=['key1','key2']))  print(pd.merge(left1,right1,how='outer',on=['key1','key2']))  print(pd.merge(left1,right1,how='left',on=['key1','key2'])) |

## Joining

Joining is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame.

|  |
| --- |
| import pandas as pd  leftJ = pd.DataFrame({'A': ['A0', 'A1', 'A2'],                       'B': ['B0', 'B1', 'B2']},                        index=['K0', 'K1', 'K2'])  rightJ = pd.DataFrame({'C': ['C0', 'C2', 'C3'],                      'D': ['D0', 'D2', 'D3']},                        index=['K0', 'K2', 'K3'])  print(leftJ.join(rightJ, how='outer')) #how default is inner |

# Operations

|  |
| --- |
| import pandas as pd  df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':['abc','def','ghi','wxyz']})  print(df.head())  print(df['col2'].unique())  print(df['col2'].nunique())#count of unique value  #Output: 3  print(df['col2'].value\_counts())#count of value in col  #Output  #   444    2  #   555    1  #   666    1 |

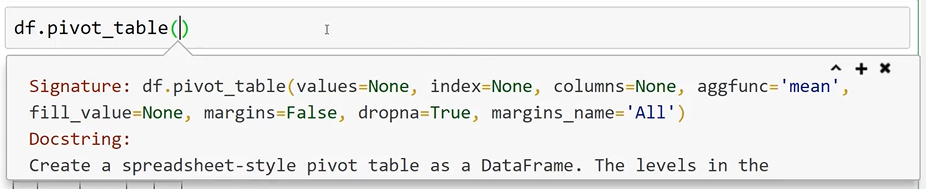
# Apply

|  |
| --- |
| df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':['abc','def','ghi','wxyz']})  def time2(x):      return x\*2  #customer function  res = df['col1'].apply(time2)  print(res)  #built in function  print(df['col3'].apply(len))  #lamda expression  print(df['col1'].apply(lambda x: x\*2)) |

# Sorting and IsNull

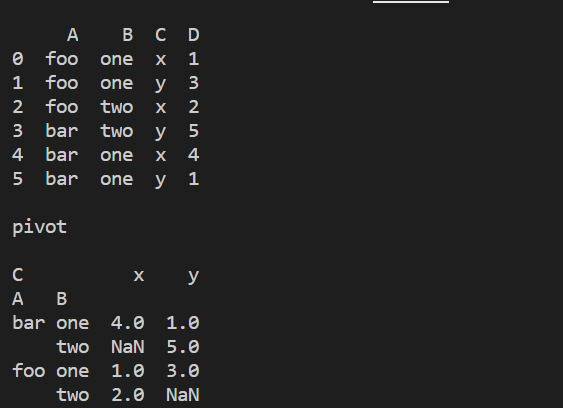
|  |
| --- |
| df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':['abc','def','ghi','wxyz']})  print(df.sort\_values(by='col2',axis=0))  print(df.isnull()) |

# Pivot



|  |
| --- |
| import pandas as pd  data = {'A':['foo','foo','foo','bar','bar','bar'],       'B':['one','one','two','two','one','one'],         'C':['x','y','x','y','x','y'],         'D':[1,3,2,5,4,1]}  df = pd.DataFrame(data)  print(df)  pv = df.pivot\_table(values='D',index=['A', 'B'],columns=['C'])  print(pv) |

Output



# Data Input and Output

* CSV
* Excel
* HTML
* SQL

# CSV

|  |
| --- |
| import pandas as pd  #read  df = pd.read\_csv(r'D:\ShareData\TCS.csv')  print(df)  #Wrire  df.to\_csv('eg.csv',index= False)  df2 = pd.read\_csv(r'eg.csv')  print(df2) |

# Excel

We need to install package xlrd: *pip install xlrd*

|  |
| --- |
| import numpy as np  import pandas as pd  #import openpyxl  df = pd.read\_excel(r'D:\ShareData\TCS\_xls.xlsx',sheet\_name='TCS')  print(df)  df.to\_excel('TCS1.xlsx',sheet\_name='TCS') |

## HTML

Install these lib

* lxml
* html5lib
* BeautifulSoup4

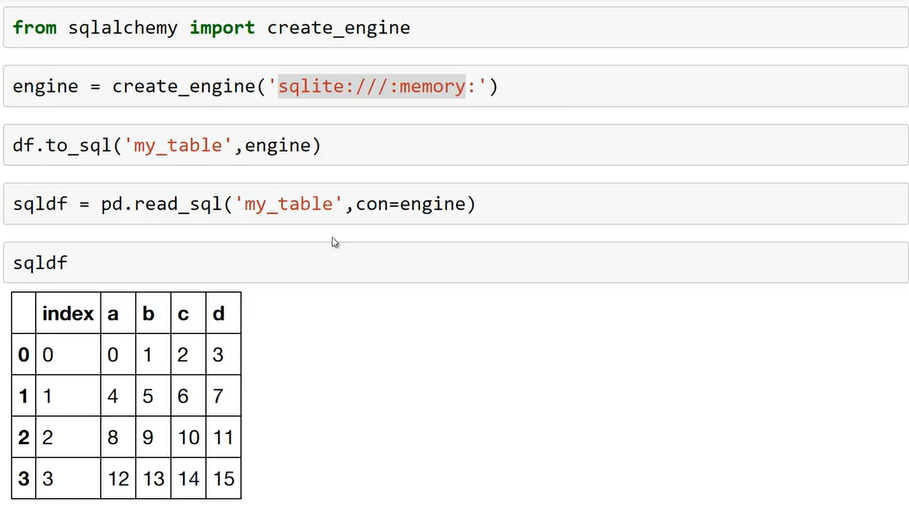
|  |
| --- |
| import pandas as pd  df = pd.read\_html('https://www.moneyworks4me.com/best-index/nse-stocks/top-nifty50-companies-list/')  print(df[0].head())#heads get first 5 records only |

## SQL Server

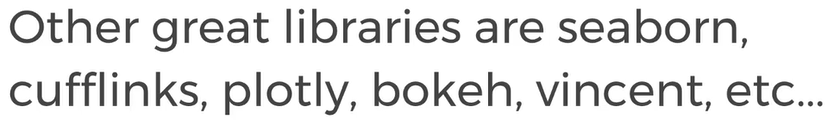
Install package *pyodbc*

|  |
| --- |
| import pandas as pd  import pyodbc  conn = pyodbc.connect('Driver={SQL Server};'                        'Server=10.2.118.163\MSSQL2016;'                        'Database=GLP\_CMP;'                        'UID=sa; PWD=abcd@1234;')  query = 'Select top 10 PlayerId,Title,FirstName,LastName from [dbo].[tPlayer](nolock)'  df = pd.read\_sql(query, sql\_conn)  print(df) |

## SQLite



Visualization and MatPlotlib



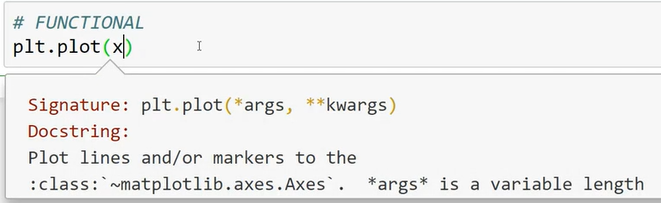
# MatPlotLib



## Install

*Pip install matplotllib*

## Functional Method of plotting

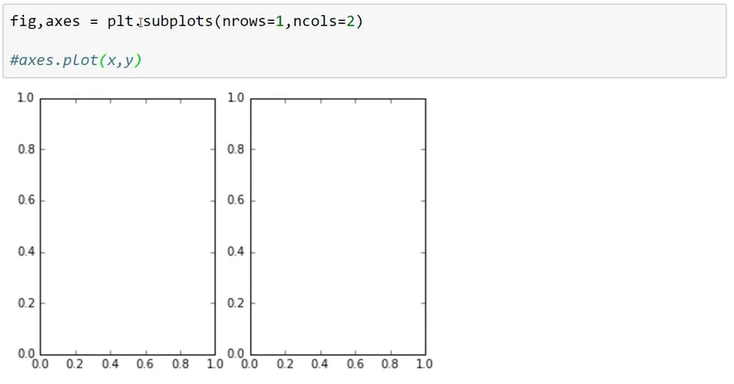


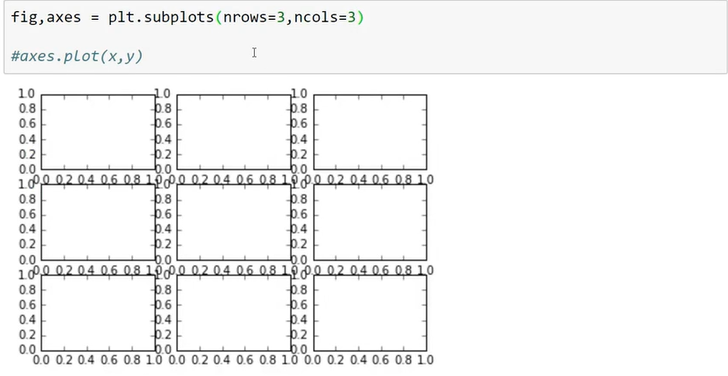
|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  x = np.linspace(0, 5, 11)  y = x \*\* 2  #Functional Method  plt.plot(x,y)  plt.xlabel('X Label')  plt.ylabel('Y Label')  plt.title('Title')  #plt.show()  #subplot  plt.subplot(1,2,1)  plt.plot(x,y,'r')  plt.subplot(1,2,2)  plt.plot(y,x,'b')  plt.show() |

## Oops Method

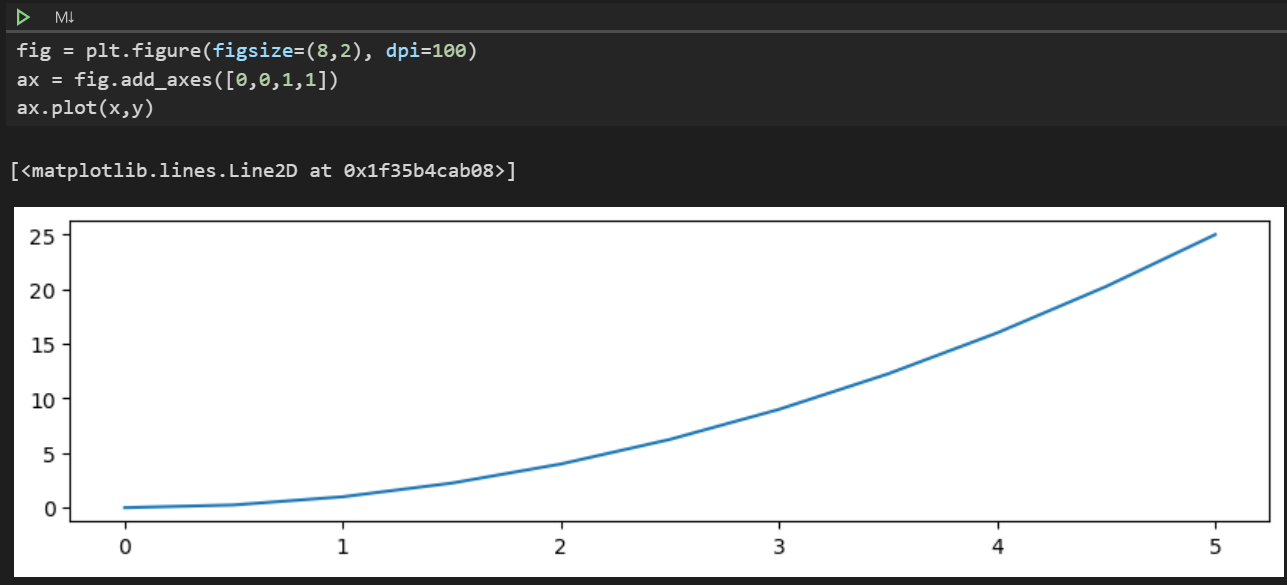
|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  x = np.linspace(0, 5, 11)  y = x \*\* 2  fig = plt.figure()  # Add set of axes to figure  axes = fig.add\_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (range 0 to 1)  # Plot on that set of axes  axes.plot(x, y, 'b')  axes.set\_xlabel('Set X Label') # Notice the use of set\_ to begin methods  axes.set\_ylabel('Set y Label')  axes.set\_title('Set Title')  #Inner plot  fig1 = plt.figure()  axes1 = fig1.add\_axes([0.1, 0.1, 0.8, 0.8])  axes2 = fig1.add\_axes([0.35, 0.15, 0.5, 0.4])  axes1.plot(x,y,'r')  axes1.set\_title('large Canvas')  axes2.plot(y,x,'b')  axes2.set\_title('Small Canvas')  plt.show() |

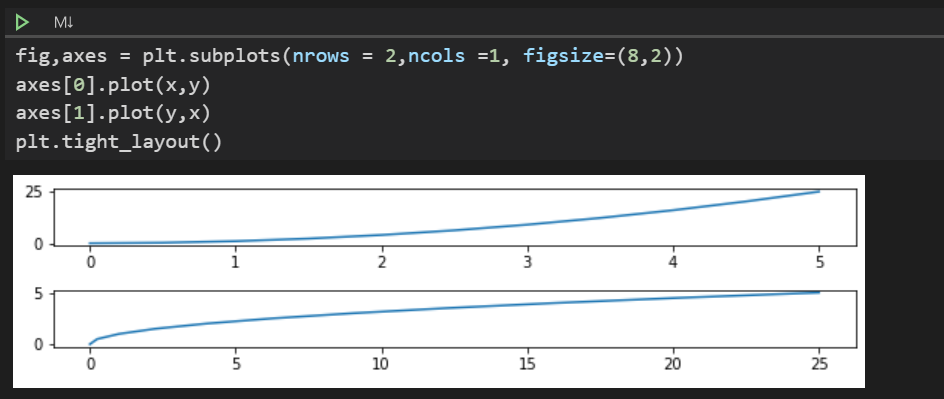
### Subplots



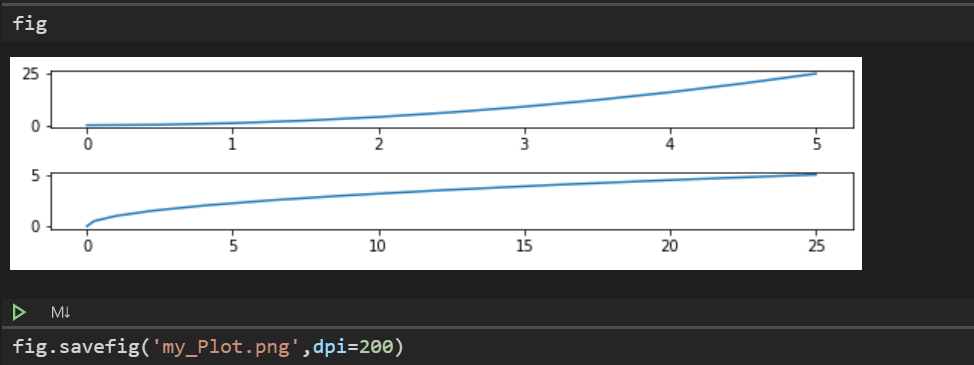
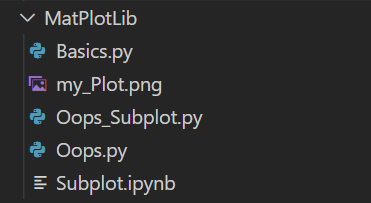


### Figure Size and DPI

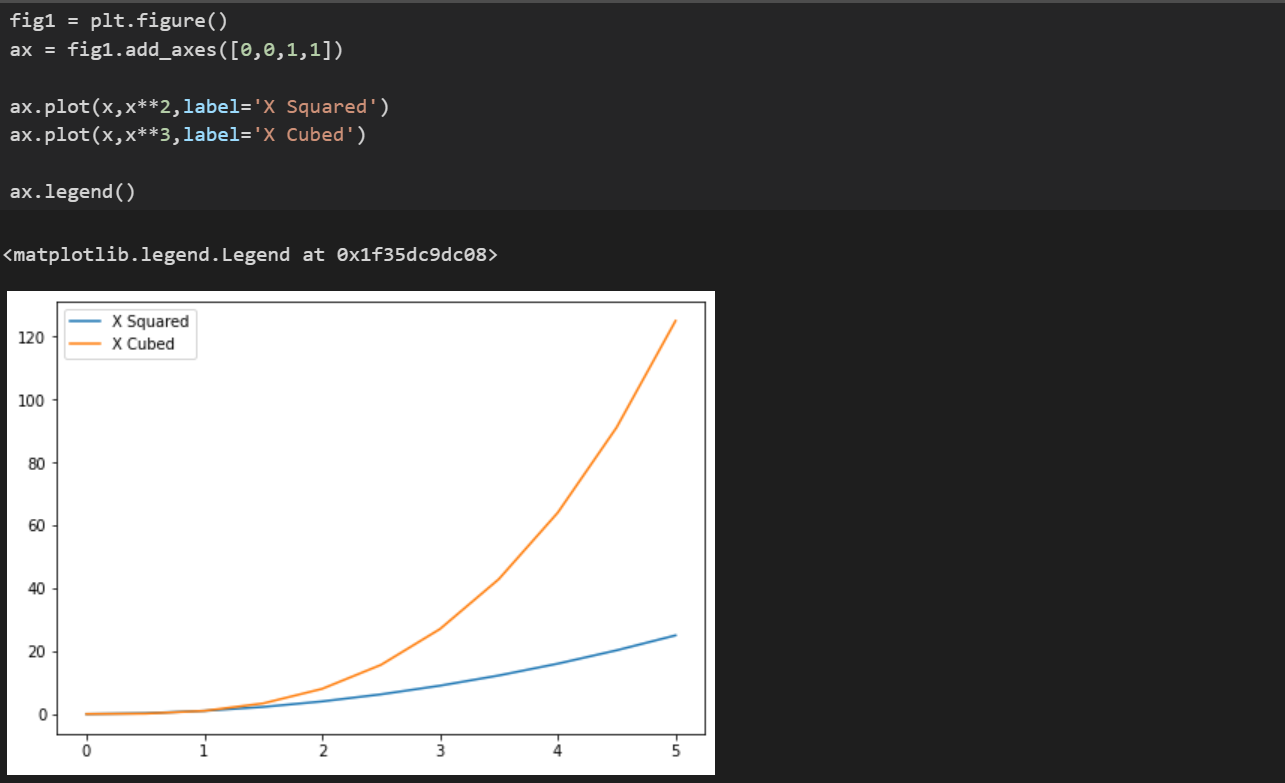


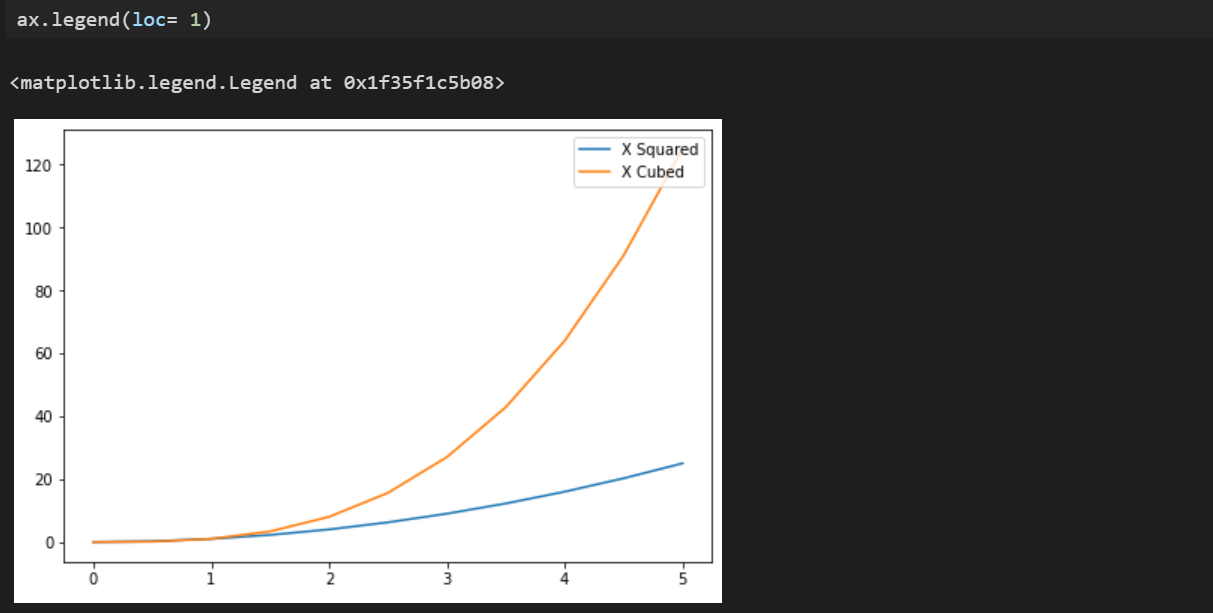


### Save Figure

### Legends





### Plot Appearance

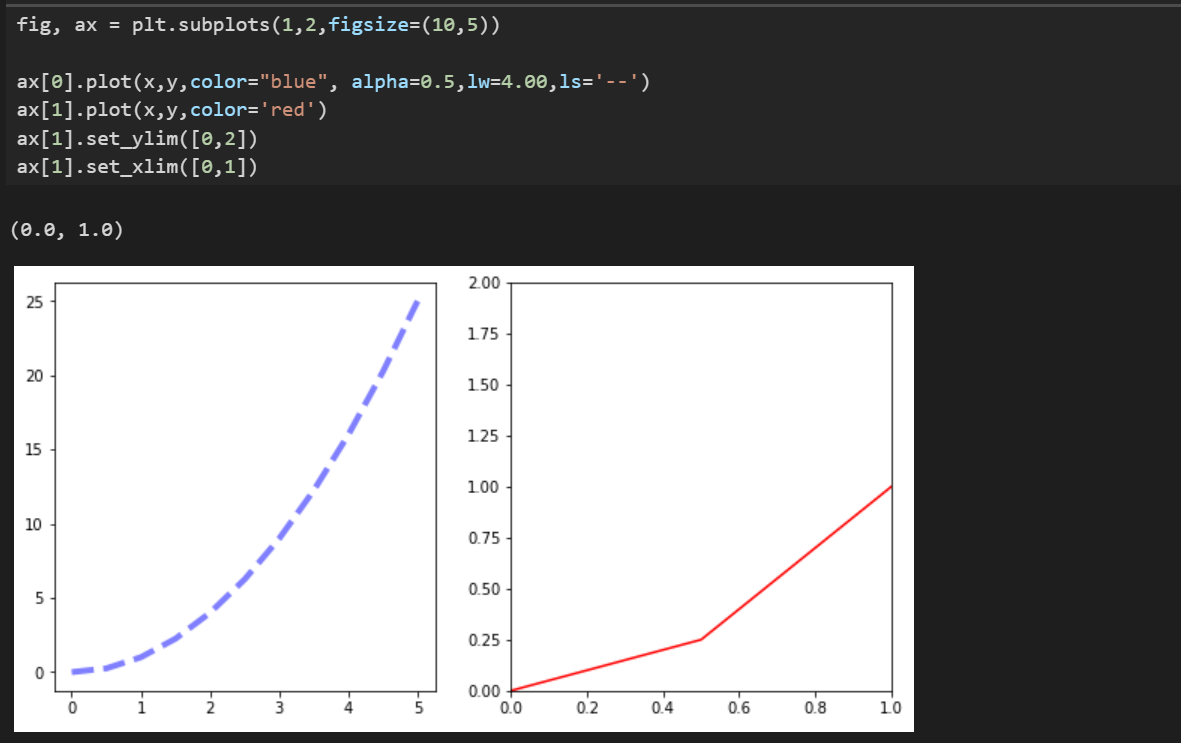
* Color
* Linewidth
* Alpha
* LineStyle
* Marker
* Marker size
* Markerfacecolor
* Marker edge width
* Marker edge color

Refer document project. It has lot of option.



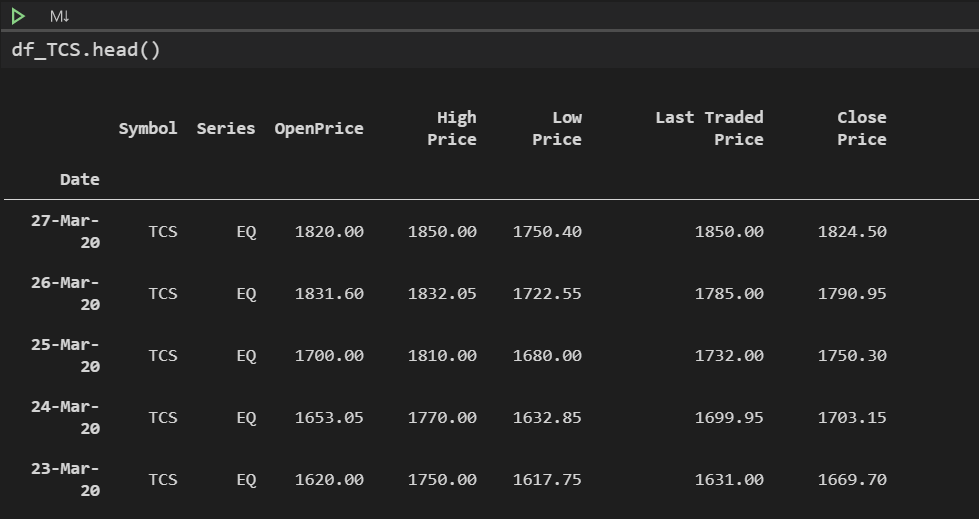
### Plot Range

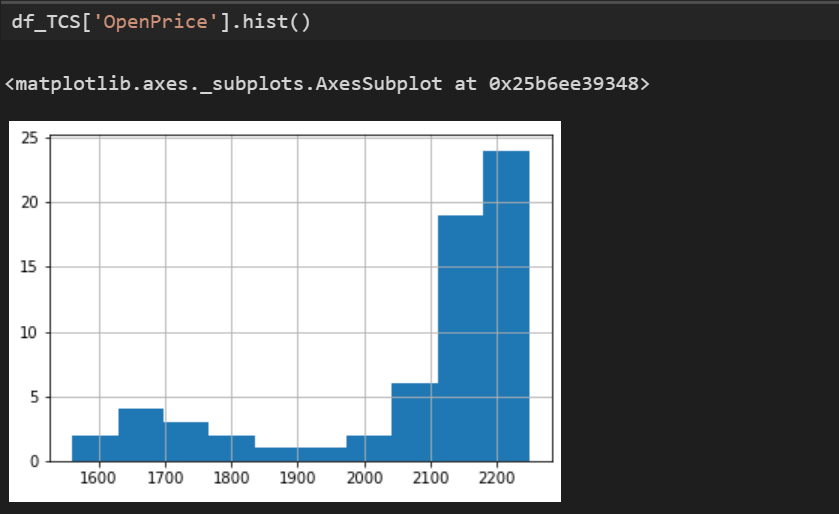
We can configure the ranges of the axes using the set\_ylim and set\_xlim methods in the axis object, or axis ('tight') for automatically getting "tightly fitted" axes ranges:



# Pandas Visualization

Create a dataframe and directly call .hist() method on dataframe column to plot the chart





## Call the Style

There are multiple style in pandas plotting. Below are some examples.

|  |
| --- |
| import matplotlib.pyplot as plt  plt.style.use('ggplot')  plt.style.use('bmh')  plt.style.use('dark\_background')  plt.style.use('fivethirtyeight') |

## Plot Types

* df\_Rel[' Close Price'].plot.area()
* df.plot.barh()
* df.plot.density()
* df.plot.hist
* df.plot.line
* df.plot.scatter
* df.plot.bar
* df.plot.box
* df.plot.hexbin
* df.plot.kde
* df.plot.pie

# Time Series

Import dataframe with parameter index\_col='Date',parse\_dates=True

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  df\_TCS = pd.read\_csv(r'D:\ShareData\TCS\_New.csv',index\_col='Date',parse\_dates=True).sort\_values(by='Date', )  output |

## Plot

|  |
| --- |
| df\_TCS['LTP'].plot()  df\_TCS['Quantity'].plot()  df\_TCS['LTP'].plot(figsize=(12,8), title='TCS)  plt.ylabel('Close Price')  plt.xlabel('Overwrite Date Index') |

## Plot Formatting

### X Limits and Y Limits

Setting limit to data for plotting graph.

|  |
| --- |
| **df\_TCS['LTP'].plot(figsize=(12,8),title='TCS',xlim=['2020-02-01','2020-03-27'],ylim=[1500,2000]** , ls='--', c='r'**)** |

### X Ticks

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import matplotlib.dates as dates  # define key value for specific column  idx = df\_TCS.loc['2020-02-01':'2020-03-27'].index  stock = df\_TCS.loc['2020-02-01':'2020-03-27']['LTP']  fig, ax = plt.subplots()  ax.plot\_date(idx, stock,'-')  fig.autofmt\_xdate()  plt.tight\_layout()  # Grids  ax.yaxis.grid(True)  ax.xaxis.grid(True)  plt.show() |

### Major and Minor Axis

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import matplotlib.dates as dates  # define key value for specific column  idx = df\_TCS.loc['2020-02-01':'2020-03-27'].index  stock = df\_TCS.loc['2020-02-01':'2020-03-27']['LTP']  fig, ax = plt.subplots()  ax.plot\_date(idx, stock,'-')  # Major Axis  ax.xaxis.set\_major\_locator(dates.MonthLocator())  ax.xaxis.set\_major\_formatter(dates.DateFormatter('\n\n\n\n%Y--%B'))  # Minor Axis  ax.xaxis.set\_minor\_locator(dates.WeekdayLocator())  ax.xaxis.set\_minor\_formatter(dates.DateFormatter('%d'))  fig.autofmt\_xdate()  plt.tight\_layout()  # Grids  ax.yaxis.grid(True)  ax.xaxis.grid(True)  plt.show() |

Data Source

# Pandas Data Reader

It auto reads data from Google or Yahoo API.

Install:

pip install pandas-datareader

pip install datetime

|  |
| --- |
| import pandas\_datareader.data as web  import datetime as dt  start = dt.datetime(2018,1,1)  end = dt.datetime(2020,1,1)  facebook = web.DataReader('FB','yahoo',start,end)  print(facebook.head()) |

# Quandl

Create an account in Quandl.com and generate ur API key.

|  |  |
| --- | --- |
| Install | pip install quandl |
| API call for 6month data | quandl.get("BSE/BOM532281", authtoken="NNVwxTvRt\_GdzLtj6Zzt", start\_date="2019-09-30") |
| List of all code for BSE | <https://www.quandl.com/api/v3/databases/>BSE/metadata?api\_key =NNVwxTvRt\_GdzLtj6Zzt |
| API KEY | API KEY: NNVwxTvRt\_GdzLtj6Zzt |

|  |
| --- |
| import quandl as q  import matplotlib.pyplot as plt  hcl\_EOD = q.get("BSE/BOM532281", authtoken="<API\_KEY>", start\_date="2019-09-30")  print(hcl\_EOD.head())  hcl\_EOD['Open'].plot()  plt.show() |

Pandas with Time Series Data

# Datetime Index

## DateTime

Install: *pip install datetime*

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from datetime import datetime  my\_year = 1987  my\_month = 7  my\_day = 17  my\_hour = 13  my\_minute = 30  my\_second = 15  my\_date\_time = datetime(my\_year,my\_month,my\_day,my\_hour,my\_minute,my\_second)  print(my\_date\_time) |

## DateTime Index

### Create

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from datetime import datetime  first\_two = [datetime(2016, 1, 1), datetime(2016, 1, 2)]  print(first\_two)  dt\_ind = pd.DatetimeIndex(first\_two) #create  print(dt\_ind)  data = np.random.randn(2,2)  print(data)  cols = ['A','B']  df = pd.DataFrame(data,dt\_ind,cols)  print(df) |

### Operation

|  |
| --- |
| ##Index  print(df.index)  print(df.index.argmax())  print(df.index.max()) |

# Time Resampling

Aggregating date based on financial groups. Like take sum of all closes based on quarter or monthly.

* Read dataframe from source (CVS in this case).
* Format and assign date column as Index
* Perform resampling based on rule you require. List of rule is given below.

|  |  |  |  |
| --- | --- | --- | --- |
| Alias | Description | Alias | Description |
| B | business day frequency | QS | quarter start frequency |
| C | custom business day frequency (experimental) | BQS | business quarter start frequency |
| D | calendar day frequency | A | year-end frequency |
| W | weekly frequency | BA | business year end frequency |
| M | month end frequency | AS | year start frequency |
| SM | semi-month end frequency (15th and end of month) | BAS | business year start frequency |
| BM | business month end frequency | BH | business hour frequency |
| CBM | custom business month end frequency | H | hourly frequency |
| MS | month start frequency | T, min | minutely frequency |
| SMS | semi-month start frequency (1st and 15th) | S | secondly frequency |
| BMS | business month start frequency | L, ms | milliseconds |
| CBMS | custom business month start frequency | U, us | microseconds |
| Q | quarter end frequency | N | nanoseconds |
| BQ | business quarter endfrequency |  |  |

* After getting sampling value plot the graph

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from datetime import datetime  #Step1 Getting Data  #There are two method of getting Financial data.  #direct method of defing Index and this faster way but less Controll  df\_sh = pd.read\_csv(r'D:\ShareData\TCS.csv',index\_col='Date',parse\_dates=True)  #Longer and Controlled Method of defining idex  df = pd.read\_csv(r'D:\ShareData\TCS.csv')  #Step 2 Convert datetime format and assign Index  #Method 1  df['Date'] = pd.to\_datetime(df['Date'])  #Method 2  #df['Date'] = df['Date'].apply(pd.to\_datetime)  print(df.info())  #Step 2.1 Assign index  df.set\_index('Date',inplace=True)  #Step 3 Resampling data as required  #Using In built functions  print(df.resample(rule='M').mean())  print(df.resample(rule='M').max())  print(df['TotalTradedQuantity'].resample(rule='M').sum())  #Customer Method  def first\_day(entry):      #Returns the first instance of the period, regardless of samplling rate.      return entry[0]  print(df.resample(rule='M').apply(first\_day))  #Step 4 Plotting Graph  df['OpenPrice'].resample('M').mean().plot(kind='bar')  plt.title('Monthly Mean Open Price for TCS')  plt.show() |

# Time Shift

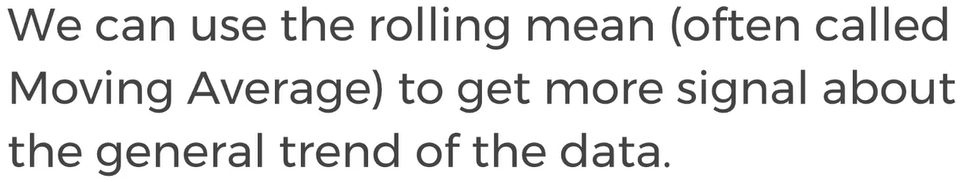
Shifting row up or down the index.

|  |
| --- |
| import numpy as np  import pandas as pd  df = pd.read\_csv(r'D:\ShareData\TCS.csv',index\_col='Date',parse\_dates=True).sort\_values(by='Date', ascending=True)  print(df)  #Period  print(df.head())  print(df.tail())  print(df.shift(periods=1).head())  print(df.shift(periods=1).tail())  print(df.head())  print(df.tail())  print(df.shift(periods=-1).head())  print(df.shift(periods=-1).tail())  #tShift  print(df.tshift(periods=1,freq='M').head()) |

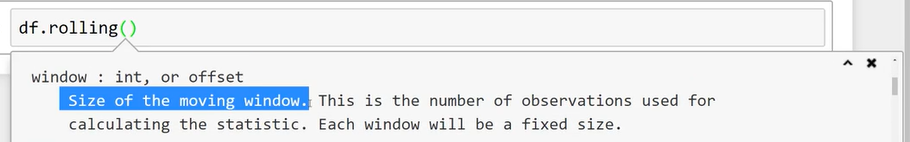
# Pandas Rolling and Expanding

## Rolling

Taking avg or mean of specific number of period. Example 7 day moving avg.







## Expanding

It basically take avg of all the data points available, continuously, and last value is avg of every data contained in data frame column. Using this we can find overall trend.

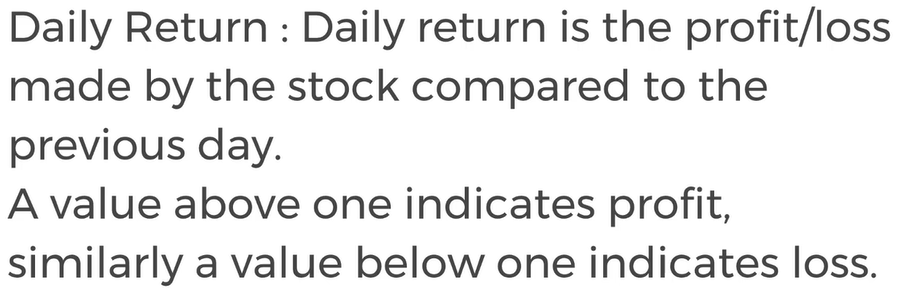


## Example: Bollinger Band

Data points required for plotting Bollinger band



# Daily Percentage Change



* Also known as Daily return
* This defines percent gain or percent loss if we sold our stock next day.
* If price is stable then return trends to be close to zero, which mean stock is less volatile.
* And less volatile or narrow distributed stock are less risky and have low return.

# Cumulative Returns

