\_\_new\_\_ >> \_\_init\_\_ ---Execution order

\_\_new\_\_ - -- object instance creation

\_\_Initi\_\_ -- for object variable initialization

Python packaging

neural network

Excellent knowledge of OOPS programming, file system, database access.

ETL transformation through Python.

Learn Python with YouTube

<https://www.youtube.com/watch?v=QXeEoD0pB3E&list=PLsyeobzWxl7poL9JTVyndKe62ieoN-MZ3&index=1>

<https://www.programiz.com/python-programming/property>

<https://www.programiz.com/python-programming/ide>

**Download Python**

<https://www.python.org/downloads/>

**Download Python IDE**

Default IDE for Python is **VS Code**, **IDLE**. But **PyCharm** is most famous IDE.

<https://www.jetbrains.com/pycharm/download/#section=windows>

Download **community version** instead of professional version.

**Python setup and environment variable**

<https://projects.raspberrypi.org/en/projects/using-pip-on-windows/5>

**What is Artificial intelligence?**

<https://www.youtube.com/watch?v=5hNK7-N23eU>

<https://www.youtube.com/watch?v=xtOg44r6dsE>

<https://www.youtube.com/watch?v=I7NrVwm3apg>

**Try-** <https://www.youtube.com/watch?v=vq2nnJ4g6N0>

A machine with the ability to perform cognitive functions such as perceiving, learning, reasoning and solve problems are deemed to hold an artificial intelligence.

Introduction to AI Levels

**Narrow AI**: A artificial intelligence is said to be narrow when the machine can perform a specific task better than a human. The current research of AI is here now

**General AI**: An artificial intelligence reaches the general state when it can perform any intellectual task with the same accuracy level as a human would

**Strong AI**: An AI is strong when it can beat humans in many tasks

Type of Artificial Intelligence

Artificial intelligence can be divided into three subfields:

* Artificial intelligence
* Machine learning
* Deep learning

**AI and Python: Why?**

Python has Prebuilt Libraries like **Numpy** for scientific computation, **Scipy** for advanced computing and **Pybrain** for machine learning (Python Machine Learning) making it one of the best languages For AI.

Python developers around the world provide comprehensive support and assistance via forums and tutorials making the job of the coder easier than any other popular languages.

Python is platform Independent and is hence one of the most flexible and popular choice for use across different platforms and technologies with the least tweaks in basic coding.

NumPy is used as a container for generic data comprising of an N-dimensional array object, tools for integrating C/C++ code, Fourier transform, random number capabilities, and other functions.

Another useful library is **pandas**, an open source library that provides users with easy-to-use data structures and analytic tools for Python.

# Framework List

|  |  |
| --- | --- |
| **Module** | **Description** |
| **Pytest** | Testing Framework |
| **NumPy** | Numerical Python--Data Analysis,Data Sceience,written in C language. |
| **Panda** | Panel Data--ETL, Huge data processing |
| **matplotlib** | Data Visualization, ETL reporting |
| **Scipy** | statistical analysis ,Advance Computing |
| **TensorFlow, Keras, Scikit-learn** | Machine Learning |
| **NLTK, spaCy** | Natural language processing |

# What Is The Difference Between Data Science And Machine Learning?

One of the most common confusions arises among the modern technologies such as artificial intelligence, machine learning, big data, data science, deep learning and more.

**Data Science**

In simple words, data science is the processing and analysis of data that you generate for various insights that will serve a myriad of business purposes. For instance, when you have logged in on Amazon and browsing through a few products or categories, you are generating data. This data will be used by a data scientist at the backend to understand your behaviour and push you retargeted advertisements and deals to get you purchase what you browsed. This is one of the simplest implementations of data science and it keeps getting more complex in terms of concepts like cart abandonment and more.

Data science involves the processes of

Data extraction

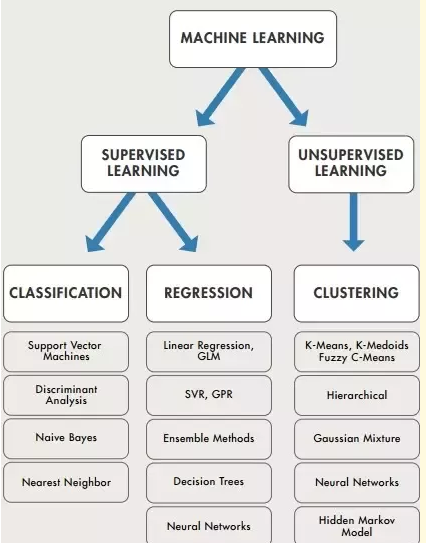
Data Cleansing

Analysis

Visualization

And actionable Insights generation

Tons of insights lie unnoticed in massive chunks of data and it is data science that sheds new light on areas like customer behaviour, operational shortcomings, supply-chain cycles, predictive analysis and more. Data science is crucial for companies to retain their customers and stay in the market.



## Machine Learning

For simple comprehension, understand that machine learning is part of data science.

It draws aspects from statistics and algorithms to work on the data generated and extracted from multiple resources. What happens most often is data gets generated in massive volumes and it becomes totally tedious for a data scientist to work on it. That is when machine learning comes into action. Machine learning is the ability given to a system to learn and process data sets autonomously without human intervention. This is achieved through complex algorithms and techniques like regression, supervised clustering, naïve Bayes and more.

### Types of Machine Learning (ML)

1.supervised

2.unsupervised.

### Supervised machine learning algorithms

This is the most commonly used machine learning algorithm. It is called supervised because the process of algorithm learning from the training dataset can be thought of as a teacher supervising the learning process.

Mainly supervised leaning problems can be divided into the following two kinds of problems −

Classification

A problem is called classification problem when we have the categorized output such as “black”, “teaching”, “non-teaching”, etc.

Regression

A problem is called regression problem when we have the real value output such as “distance”, “kilogram”, etc.

Decision tree, random forest, knn, logistic regression are the examples of supervised machine learning algorithms.

## Unsupervised machine learning algorithms

As the name suggests, these kinds of machine learning algorithms do not have any supervisor to provide any sort of guidance.That is why unsupervised machine learning algorithms are closely aligned with what some call true artificial intelligence.

Unsupervised learning problems can be divided into the following two kinds of problem

Clustering − in clustering problems, we need to discover the inherent groupings in the data. For example, grouping customers by their purchasing behavior.

Association − A problem is called association problem because such kinds of problem require discovering the rules that describe large portions of our data. For example, finding the customers who buy both x and y.

K-means for clustering, Apriori algorithm for association are the examples of unsupervised machine learning algorithms.

**NLP (Natural language processing)** is simply the part of AI that has to do with language (usually written).

**Deep learning** is one kind of machine learning that’s very popular now. It involves a particular kind of mathematical model that can be thought of as a composition of simple blocks (function composition) of a certain type, and where some of these blocks can be adjusted to better predict the final outcome.

# Variable

## Integer Variable

X=10

Y=20

X+Y o/p=30

X/Y

X%Y o/p=

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| \*, % | Multiplication, Modulo | 27 % 7 |
| Result: 6 |
| \*\* | Exponentiation | 10 \*\* 3 |
| Result: 1000 |
| or, and, not | Boolean Or, Boolean And, Boolean Not | (a or b) and c |
| <, <=, >, >=, !=, == | The usual comparison operators |  |
| +, - | Addition, Subtraction | 3-Oct |

y=20  
x=10  
if (x==y **or** x==10):  
 print(**'both are equal'**) #This would be executed  
else:  
 print(**"both are not equal"**)

### most recent output value

The most recent output value is automatically stored by the interpreter in a special variable with the name "\_". So we can print the output from the recent example again by typing an underscore after the prompt:

The underscore variable is only available in the Python shell. It's NOT available in Python scripts or programs.

## String Variable

Name=’MD KHALID ANWAR’

### Length of string

**len(**StringVariable**)**

### Conversion to String

str function

str(tuple\_variable)

str(list\_varible)

### **Concatenation of String**

x1=**"Aatikah"**x2=**"Khalid"**print(x1+**" "**+x2)

### **Lower or Upper case**

x=**"MD Khalid Anwar"**print(x.lower())  
print(x.upper())

**Multiplication of String:**

x=”hI”

Print(x \* 2) # o/p=hihi

**PlaceHolder:**

%s ----for String

%d ---- for integer

Str1=”My Name is %s”

Str1%(“Khalid Anwar”) o/p=My Name is Khalid Anwar

Str2=”My age is %d”

Str2%(35) o/p=My age is 35

Str3=”My Name is %s and age is %d”

Str3%(“Khalid”,35) o/p= My Name is Khalid and age is 35

#### \_\_name\_\_ keyword

It is built-in variable

Since there is no main() function in Python, when the command to run a python program is given to the interpreter, the code that is at level 0 indentation is to be executed.

If the source file is executed as the main program, the interpreter sets the \_\_name\_\_ variable to have a value “\_\_main\_\_”. If this file is being imported from another module, \_\_name\_\_ will be set to the module’s name.\_\_name\_\_ is a built-in variable which evaluates to the name of the current module. Thus it can be used to check whether the current script is being run on its own or being imported somewhere else by combining it with if statement, as shown below.

If same file is executed then \_\_name\_\_=\_\_manin\_\_

If file is imported then \_\_name\_\_=module name

# File1.py

print(**"File1 \_\_name\_\_ ="**+\_\_name\_\_)  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
 print(**"File1 is being run directly"**)  
else:  
 print(**"File1 is being imported"**)

Output :

File1 \_\_name\_\_ = \_\_main\_\_

File1 is being run directly

# File2.py

import File1 #all codes except function will be executed first and other code in this file

print(**"File2 \_\_name\_\_ ="**+\_\_name\_\_)  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
 print(**"File2 is being run directly"**)  
else:  
 print(**"File2 is being imported"**)

Output :

File1 \_\_name\_\_ =kk.home.File1

File1 is being imported

File2 \_\_name\_\_ =\_\_main\_\_

File2 is being run directly

## Array Variable

Python does not have built-in support for Arrays, but [Python Lists](https://www.w3schools.com/python/python_lists.asp) can be used instead.

Name=’KHALID ANWAR’

Name[0] o/p=K ;it reads from begining

Name[-1] o/p=R ; it reads from end with index with 1 ;

Name[startIndex:EndIndex]

Name[2:3] o/p=A

Name[7:] o/p=ANWAR ;It reads all string starting from 2

Name[:2] o/p=KH-- it reads from beginning till end index

### format() function

Syntax : { } .format(value)

Formatters work by putting in one or more replacement fields and placeholders defined by a pair of curly braces { } into a string and calling the str.format().

print ("{}, A computer science portal for geeks.".format("GeeksforGeeks"))

str = "This article is written in {}"

print (str.format("Python"))

### Using Multiple Formatters

Syntax : { } { } .format(value1, value2)

print ("This is {} {} {} {}".format("one", "two", "three", "four"))

### Using Index

When placeholders { } are empty, Python will replace the values passed through str.format() in order.

The values that exist within the str.format() method are essentially tuple data types and each individual value contained in the tuple can be called by its index number, which starts with the index number 0.

These index numbers can be passes into the curly braces that serve as the placeholders in the original string.

# Reverse the index numbers with the

# parameters of the placeholders

print("{1} love {0}!!".format("GeeksforGeeks","Geeks"))

## List versus Array

|  |  |
| --- | --- |
| **List** | **Array** |
| Can consist of elements belonging to different data types | Only consists of elements belonging to the same data type |

## List Variable

Lst1=**[**“abc1”,”abc2”,30,”def1”**]**

Lst1[1] o/p=abc2

Lst1[1:3] o/p=[abc2,30]

Lst1.**append**(“hello”) -----appending value at last index

Lst1.append(33)

Lst1[0]=abc ---updating value

**del** lst1[1] ---this will delete at index 1

**len**(lst1) -- size of list

lst1 + lst2 -- adding two list

**max**(lst1) -- maximum value in array list

**min**(lst1) -- minimum value in array list

li1.**insert**(index,value]

ls1.**extend**(ls2) --ls1 will become large list(ls1+ls2) and ls2 will not be changed

li1.**remove**(value) ---deleted given value from list

li1.**pop**(index) ---deleted at given index from list

Example

#Sort

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

numbers.**sort**(reverse=True)

numbers = set(numbers)

print(numbers)

### Shallow Copy

#A shallow copy creates a new object but inserts references to the objects found in the original.

import copy

L1 = [[1, 2, 3], [4, 5, 6]]

L2 = copy.copy(L1)

L1[0][0] = 99  # Modify an element in the original list

print("Original List:", L1)

print("Shallow Copy:", L2)

## Deep Copy

# A deep copy creates a completely independent copy of the original object and all its nested objects.

import copy

L1 = [[1, 2, 3], [4, 5, 6]]

L2 = copy.deepcopy(L1)

L1[0][0] = 99  # Modify an element in the original list

print("Original List:", L1)

print("Deep Copy:", L2)

### Built-in Functions

**Min**() return minimum element of a given list

**Max**() return maximum element of a given list

**len**() Returns the length of the list

**Sum**() Returns the sum of all the numbers in the list

Append() Adds a new item to the end of the list

**Clear**() Removes all items from the list

**Count**() Returns the number of occurrences of a particular item in a list

**Index**(Value) Returns the index of a specific element of a list

### Slicing

In many programming languages it can be quite tough to slice a part of a string and even tougher, if you like to address a "subarray". Python makes it very easy with its slice operator. Slicing is often implemented in other languages as function with possible names like "**substring**", "gstr" or "substr".

str = "Python is great"

first\_six = str[0:6]

print(first\_six) #o/p=='Python'

starting\_at\_five = str[5:]

print(starting\_at\_five) #o/p=='n is great'

a\_copy = str[:]

without\_last\_five = str[0:-5]

print(without\_last\_five) #o/p=='Python is '

### Searching in List or string

abc = ["a","b","c","d","e"]

print("a" in abc) #o/p==True

print("a" not in abc) #o/p==False

print("e" not in abc) #o/p==False

print("f" not in abc) #o/p==True

str = "Python is easy!"

print("y" in str) #o/p==True

print("x" in str) #o/p==False

### Sublist

person = [["Marc","Mayer"],["17, Oxford Str", "12345","London"],"07876-7876"]

print(person[0]) #o/p== ['Marc', 'Mayer']

print(person[0][0]) #o/p==Marc

print(person[0][1]) #o/p==Mayer

print(person[1][0]) #o/p==17, Oxford Str

## Dictionary

Key:ValuePair

dic=**{**"key1":100,"key2":200,"key3":300**}**

dic[“key”] ###o/p:100

dic1={1:100,2:200}

dic1[1] ###o/p:100

dic2={1:100,2:200,1:300} --cannot have duplicate key; if yes then override with latest value

dic2[1] ###o/p: 300

**del** dic[1]

**len**(dic)

dic.items() //need to recheck this

--

Value of dictionary could be either string, integer, boolean, a list, another dictionary

dictionary = {

1: 'hello',

'two': True,

'3': [1, 2, 3],

'Four': {'fun': 'addition'},

‘Six’:[

{'fun1': 'addition'},

{'fun2': 'addition'},

]

5.0: 5.5

}

## Tuples

**(**Same as List except cannot update value**)**

it is immutable. i.e. a tuple **cannot be changed** in any way once it has been created. A tuple is defined analogously to lists, except that the set of elements is enclosed in parentheses instead of square brackets.

dup1=**(**“abc1”,”abc2”,30,”def1”**)**

dup1[1] o/p=abc2

dup1[1]=”hello” -- throw error

**Where is the benefit of tuples?**

Tuples are faster than lists.

If you know that some data doesn't have to be changed, you should use tuples instead of lists, because this protects your data against accidental changes.

The main advantage of tuples consists in the fact that tuples can be used as keys in dictionaries, while lists can't.

**Difference between List and Tuple**

<https://www.afternerd.com/blog/difference-between-list-tuple/>

* Lists are **mutable** whereas tuples are **immutable**.

## Set

s={1,2,3,4,5,5}

Print(s) o/p: 1,2,3,4,5

*set -- built-in function*

If we want to create a set, we can call the built-in set function with a sequence or another iterable object:

>>> x = set("A Python Tutorial")

>>> x

{'A', ' ', 'i', 'h', 'l', 'o', 'n', 'P', 'r', 'u', 't', 'a', 'y', 'T'}

We can pass a list to the built-in set function, as we can see in the following:

>>> x = set(["Perl", "Python", "Java"])

>>> x

{'Python', 'Java', 'Perl'}

### List to tuple conversion

Using function *tuple()*

lst=[1,2,3,4]

Print(tuple(lst)

### tuple to list conversion

Using function *list()*

Tpl=(1,2,3,4)

Print(list(tpl)

### tuple to Set conversion

Using function *set()*

### Comparing of two sets

#### Intersection of two sets using &

Print(set\_var1 & set\_var2)

#### Non matching of two sets using ^

Print(set\_var1 ^ set\_var2)

# Conditional Statement

3 >2 (>= < <= == !=)

Single = is only for assigning value to variable age=30; 5=3(integer cannot be variable )

Indentation is more important in case of conditional statement.i.e TAB + statement should be used immediately after if condition.

## If elif else

**If**(Condition)**:**

[TAB] statement

**elif**(condition)**:**

**else:**

[TAB] statement

If(age>30 **and** age<50):

Print(“Gentle man”)

elif(age<10 or age<5):

Print(“child”)

If(age>30 **or** age<50)

Print(“Gentle man”)

elif(age<10 or age<5):

Print(“child”)

--

a=11

if a%2==0:

  print('Even')

else:

  print('odd')

## For Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

for val in sequence:

Body of for

### List iterator

lst=[1,2,3,4,5,6]

For varName in lst:

Print(varName)

### Range with default increment

Range(0,10) -- means 0 to 9. Most useful is for loop, default increment is 1

For I in range(0,10)**:**

Print(i) -- print 0 …9

### Range with defined increment

For I in range(0,11,2)**:**

Print(i) -- print 0 2 4..10 --by default is 1, in this case incrementing by 2

## Nested For Loop

For i in range(0,3)

For j in range(0,2)

Print(i\*j)

## The range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as range(**start**, **stop**, **stepSize**). step size defaults to 1 if not provided.

This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

To force this function to output all the items, we can use the function list().

print(list(range(10))) # Output: [0,1,2, 3, 4, 5, 6, 7,8,9] start point=0,increment=1

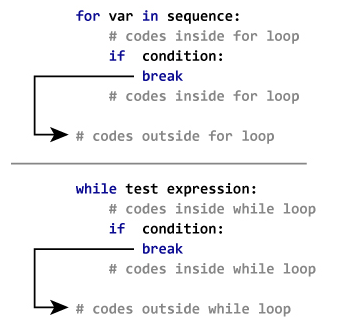
print(list(range(2, 8))) # Output: [2,3,4 ,5,6,7, 8] start point=2,increment=1

print(list(range(2, 10, 3))) # Output: [2,5, 8] start point=2,increment=3

## break/continue statement

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.



## Pass Statement

In Python programming, pass is a null statement. The difference between a comment and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored.

We generally use it as a placeholder.

Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. They cannot have an empty body. The interpreter would complain. So, we use the pass statement to construct a body that does nothing.

sequence = {'p', 'a', 's', 's'}

for val in sequence:

pass

# in function

def function(args):

pass

#in class

class example:

pass

## While Loop

C=0

while(c<10)**:**

print(c)

if(c==5):

**break**

c=c+1 ‘instead we can use --c +=1

o/p: 0 1 2 3 4 5

--

while(c<10)**:**

c=c+1

if(c>3):

**continue**

print(c)

o/p: 1 2 3

Similar to the if statement, the while loop of Python has also an optional else part. This is an unfamiliar construct for many programmers of traditional programming languages.

while condition:

statement\_1

...

statement\_n

else:

statement\_1

...

statement\_n

for i in range(1,11):

  if i==4:

    continue

  print(i)

for i in range(1,11):

  if i==4:

    pass

  print(i)

for i in range(1,11):

  if i==4:

    break

  print(i)

# String Operation

## Print half of string value

i = "Md Khalid Anwar"

n=len(i) // 2

i[:n]

Reverse String

txt = "Hello World"[::-1]

print(txt)

1234567890   (remove 0 from the number)

a="1234567890"

b=a.replace('0','')

print(b)

num=1234567890

n1=str(num).replace('0','') #convert number to string and then replace

print(int(n1)) # converting string to number and then printing

# Exception Handling

## Try except

try:

if(cc==3):

print("hello")

except:

print("something wrong")

o/p: something wrong -- As we did not declare variable cc

## try except finally

try:

  print(1/0)

  print("hello")

except:

    print("something wrong")

finally:

     print("This should be exexuted")

another Example:

Handling AttributeError exception

def abc(a,b):

  print('gg')

try:

  abc()

except(AttributeError) as e:

  print('ff',e)

finally:

  print('ddd')

**Example**

try:  
 a=10  
 b=a/0  
except Exception as e:  
 print(e)

# Comment

## Single line comment

#this is single line comment

## Multiline comment

“””

this is line1 comment

this is line2 comment

this is line3 comment

“””

## ShortCut key to comment

Ctl + / for comment or uncomment

# Function

The parameter list consists of none or more parameters. Parameters are called arguments, if the function is called. The function body consists of indented statements. The function body gets executed every time the function is called.

Parameter can be mandatory or optional. The optional parameters (zero or more) must follow the mandatory parameters.

### Create function

**def** functionName(var1,var2):

Statement1

Statement2

Statement3

def fun(x):

print(x)

Good thing is that you don’t have to specify dataType in argument

### Call function

functionName(value1,value2)

fun(“Hello”)

### Create function with return

def add(x,y):

return x+y

After return statement, there should not be any statement. if yes that code will be skipped.

sum=add(10,20) -- storing return value in variable ‘sum’

### return in Function

Function bodies can contain one or more return statement. They can be situated anywhere in the function body. A return statement ends the execution of the function call and "returns" the result, i.e. the value of the expression following the return keyword, to the caller. If the return statement is without an expression, the special value **None** is returned. If there is no return statement in the function code, the function ends, when the control flow reaches the end of the function body and the value "None" will be returned.   
Example:

def fahrenheit(T\_in\_celsius):

""" returns the temperature in degrees Fahrenheit """

return (T\_in\_celsius \* 9 / 5) + 32

for t in (22.6, 25.8, 27.3, 29.8):

print(t, ": ", fahrenheit(t))

### Optional Parameters

Functions can have optional parameters, also called default parameters. Default parameters are parameters, which don't have to be given, if the function is called. In this case, the default values are used. We will demonstrate the operating principle of default parameters with an example. The following little script, which isn't very useful, greets a person. If no name is given, it will greet everybody:

def Hello(name="everybody"):

""" Greets a person """

print("Hello " + name + "!")

Hello("Peter")

Hello()

### Docstring

The first statement in the body of a function is usually a string, which can be accessed with function\_name.\_\_doc\_\_   
This statement is called **Docstring**.   
Example:

def Hello(name="everybody"):

""" Greets a person """

print("Hello " + name + "!")

print("The docstring of the function Hello: " + Hello.\_\_doc\_\_)

The output:

The docstring of the function Hello: Greets a person

### Returning Multiple Values

A function can return exactly one value, or we should better say one object. An object can be a numerical value, like an integer or a float. But it can also be e.g. a list or a dictionary. So, if we have to return, for example, 3 integer values, we can return a list or a tuple with these three integer values. This means that we can indirectly return multiple values.

def fib\_intervall(x):

""" returns the largest fibonacci

number smaller than x and the lowest

fibonacci number higher than x"""

if x < 0:

return -1

(old,new, lub) = (0,1,0)

while True:

if new < x:

lub = new

(old,new) = (new,old+new)

else:

return (lub, new)

while True:

x = int(input("Your number: "))

if x <= 0:

break

(lub, sup) = fib\_intervall(x)

print("Largest Fibonacci Number smaller than x: " + str(lub))

print("Smallest Fibonacci Number larger than x: " + str(sup))

<https://www.python-course.eu/python3_functions.php>

### Local and Global Variables in Functions

def f():

print(s)

s = "Python"

f()

--

o/p:Python //Global Variable

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

def f():

s = "Perl"

print(s)

s = "Python"

f() //This will return local variable value

print(s) //This will return Global variable value

o/p:

Perl

Python

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

def f():

print(s)

s = "Perl"

print(s)

s = "Python"

f()

print(s)

o/p: error

The variable s is ambigious in f(), i.e. in the first print in f() the global s could be used with the value "Python". After this we define a local variable s with the assignment s = "Perl"

--

def f():

global s

print(s)

s = "dog"

print(s)

s = "cat"

f()

print(s)

o/p:

cat

dog

dog

--We made the variable s global inside of the script on the left side. Therefore anything we do to s inside of the function body of f is done to the global variable s outside of f.

### Built-in Function

**eval**(stringAsPythonCode)

**exec**(Complex- stringAsPythonCode)

**str**(ConvertAnythingToString)

**int**(IntegerConvertableString)

**float**(ConvertToFloat)

**abs**(PositiveOrNegativeNumber2Absoloutevalue)

**dir**(eitherStringOrInteger) -- gives corresponding functions

### Lambda Code

x = lambda a, b, c : a + b + c

print(x(5, 6, 2))

“”””””””””””””””””””””””””

y=lambda x:'Even' if x%2 ==0 else 'odd'

print(y(3))

# Oops

## Defining class

Class is a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.

class Parrot:

pass

## Object

object is one of instances of the class. which can perform the functionalities which are defined in the class. An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

obj = Parrot()

**self** :self represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class in python. this is just like this keyword in java. In Python, whether you have parameter or not; you must pass self as parameter variable in method

class abc:  
 age=40  
 name=**"Md Khalid Anwar"** def ageDetails(self):  
 return **f"Name--**{self.name} **age--**{self.age}**"**a=abc()  
print(a.ageDetails())

## default parameter in method

def add(x,y=10)

print(x+y)

add(1)

o/p--- 11

## arbitrary positional argument

# Arbitrary Positional Arguments

def function\_sum(\*args):

  return sum(args)

function\_sum(10,2,3,6,7,8,5, 59, 78)

## Constructors in Python

Class functions that begins with double underscore (\_\_) are called special functions as they have special meaning. Of one particular interest is the **\_ \_**init**\_ \_**() function. This special function gets called whenever a new object of that class is instantiated.

class myclass2():  
 def \_\_init\_\_(self,var1,var2): *#Constructor* self.var1=var1  
 self.var2=var2  
 def getme(self):  
 print(**f"**{self.var1} **,**{self.var2}**"**)  
*#Creating object with initialization*P=myclass2(**"Hello"**,**"Md Khalid Anwar"**)  
P.getme()  
  
class myclass():  
 def myMethod(self):  
 print(**"Hello, My Method"**)  
p=myclass()  
p.myMethod()

example1:

class abc:

  def show(self):

    aa="MD Khalid Anwar"

    print("Hello")

    return f"This is my name {aa}"

a=abc()

a.show()

example2:

self can be replaced with xyz . but it should be replaced with all self variable

class student:

  def \_\_init\_\_(xyz, name, age, gender):

    xyz.name = name

    xyz.age = age

    xyz.gender = gender

  def show(xyz):

    #return "student name" + self.name + "age" + self.age + "gender" + self.gender

    return f"student name: {xyz.name}, age is {xyz.age} and gender is {xyz.gender}"

obj1 = student("Aakash", 23, "male")

print(obj1.show())

obj2 = student("Vikash", 33, "male")

print(obj2.show())

## Inheritance

Obj.\_\_mro\_\_

class A:

  def show(self):

    print('i am in calss A')

  def A\_method(self):

    print('i am in calss A')

class B:

  def show(self):

    print('i am in calss B')

  def B\_method(self):

    print('i am in calss B')

class C(B,A):

  #pass

  def show(self):

     print('i am in calss C')

  def C\_method(self):

      print('i am in calss C')

obj1 = C()

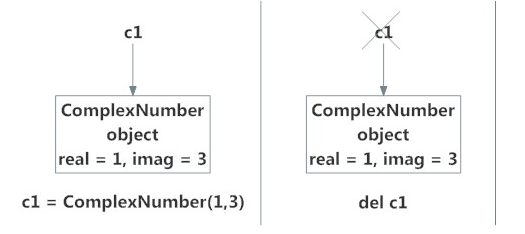
obj1.show()

print(C.\_\_mro\_\_)

obj1.A\_method()

## Garbage Collection

When we do c1 = ComplexNumber(1,3), a new instance object is created in memory and the name c1 binds with it.On the command del c1, this binding is removed and the name c1 is deleted from the corresponding namespace. The object however continues to exist in memory and if no other name is bound to it, it is later automatically destroyed.



## Inheritance

Parent and child relationship is called inheritance. All properties and functions will be part of child class.

Class child(parentClassname):

def \_\_init\_\_(self):

print(“this is child class”)

p=child()

## Encapsulation

A=”hello” --Public access modifier

\_a=”hello” --Protected

\_ \_ a=”hello” --private

class A:

  \_a = 10 # Protected

  \_\_b = 20 # private

  c = 30

   # def show(self):

  #   print('Value of A:', A.a)

  #   print('Value of B:', A.b)

obj = A()

print(obj.c)

print(A.\_a)

## Abstraction

For abstraction **abc** is package wherein **ABC** class is imported. This is default package for abstraction class implementation.

#abstraction

from abc import ABC, abstractmethod

class CAR(ABC):

  def engine\_start(self):

    print('Engine start')

  @abstractmethod

  def speed(self):

    pass

class FORD(CAR):

  def speed(self):

    print('Max speed : 100 KMPH')

class BMW(CAR):

  def speed(self):

    print('Max speed : 200 KMPH')

ford = FORD()

ford.engine\_start()

ford.speed()

bmw = BMW()

bmw.engine\_start()

bmw.speed()

## polymorphism

### Overloading

There is no overloading concept in python.

class A:  
 def show(self):  
 print(**"0 argument function"**)  
 def show(self,x):  
 print(**"1 argument function"**)  
 def show(self,x,y):  
 print(**"2 argument function"**)

Above code throw error. There is no overloading. This could be achieved with another way. In Python, \***args** and \*\***kwargs** are used to allow functions to accept an arbitrary number of arguments. These features provide great flexibility when designing functions that need to handle a varying number of inputs.

There are two special symbols to pass multiple arguments:

* \*args (Non-Keyword Arguments) stores as tupples
* \*\*kwargs (Keyword Arguments)

class B:  
 def show(self,**\***any):  
 print(**"show method"**)  
b=B();  
b.show()  
b.show(1)  
b.show(**"ss"**,1)

**Another Example**

Iterating over every argument

def print(self,\*argv):

print(“multiple argument values”,argv)  
 for a in argv:  
 print(a)

b.print(**"Hello"**,**"Mr"**,**"Khalid"**,**"Anwar"**)

\*\***kwargs**

The special syntax *\*\*kwargs* in function definitions is used to pass a variable length argument list. We use the name *kwargs* with the double star \*\*.

* A keyword argument is where you provide a name to the variable as you pass it into the function.
* It collects all the additional keyword arguments passed to the function and stores them in a dictionary.

def fun(self,\*\*kwargs):  
 for k, val in kwargs.items():  
 print(**"%s == %s"** % (k, val))

b.fun(s1=**'Geeks'**, s2=**'for'**, s3=**'Geeks'**)

**##** For s1=’Geeks’, s1 is key and ‘Geeks’ is a value. In simple words, what we assign is value and to whom we assign is key.

**Both \*args and \*\*kwargs**

def fun2(self,\*args, \*\*kwargs):  
 print(**"Positional arguments:"**, args)  
 print(**"Keyword arguments:"**, kwargs)

fun(1, 2, 3, a=4, b=5)

### Overriding

Parent method will be overridden by child class as below. It is similar to java

class Parent():  
 def \_\_init\_\_(self):  
 print("This is parent class")  
 def func(self):  
 print("this is my function")  
 def overrideMethod(self):  
 print("This is parent method")  
class child(Parent):  
 def \_\_init\_\_(self):  
 print("This is child class")  
 def childfun(self):  
 print("this is child function")  
 def overrideMethod(self):  
 print("This is Child method")  
p=child()  
p.func()  
p.childfun()  
p.overrideMethod()  
print("--------------------")  
p1=Parent()  
p1.overrideMethod()

## Generator

Problem: RAM memory utilization

#generator

import sys

def createList():

  list = []

  i = 1

  while i<200:

    list.append(i)

    i +=1

    return list

print(createList())

z = sys.getsizeof(createList())

print(z)

## Solution: Iterator

def createList():

  i =1

  while i<200:

**yield** i

    i +=1

print(createList())

x = createList()

print(next(x))

print(next(x))

print(next(x))

print(next(x))

print(next(x))

### getsizeof method

This return size of any variable.

import sys  
x=100;  
y=[1,2,3,4,5]  
name=**"Md Khalid ANwar"**list=range(100)  
print(sys.getsizeof(x))  
print(sys.getsizeof(y))  
print(sys.getsizeof(name))  
print(sys.getsizeof(list))

## Iterator – iter(value)

#iterator code

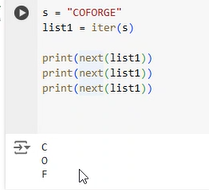
s = "COFORGE"

list1 = **iter**(s)

print(next(list1))

print(next(list1))

print(next(list1))



## List comprehension

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

even\_numbers = [x for x in numbers if x % 2 == 0]

print(even\_numbers)

Example

######List Comprehention

# Normal Code using for loop

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = []

for x in fruits:

  if "a" in x:

    newlist.append(x)

print(newlist)

#List comprehensions provide a concise way to create lists bold text

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

#         find --> For loop -->C ondition

newlist = [x for x in fruits if "a" in x]

print(newlist)

## Dictionary comprehension

new\_dict = {key\_expression: value\_expression for item in iterable if condition}

find—for loop --condition

numbers = [1, 2, 3, 4, 5]  
squared\_dict = {x: x\*\*2 for x in numbers}    
# Result: squared\_dict = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# simple Dictionary

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

squares = {}

for num in numbers:

   squares[num] = num\*\*2

print(squares)  # Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

#Dictionary  comprehention

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

squares = {num: num\*\*2 for num in numbers}

print(squares)  # Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

## any

any() method is a built-in function that returns TRUE if any of the items of a provided iterable (List, Dictionary, Tuple, Set, etc.) are true; otherwise, it returns FALSE.

## all

## Singleton design pattern

A Singleton pattern in python is a design pattern that allows you to create just one instance of a class, throughout the lifetime of a program.

* To limit concurrent access to a shared resource.
* To create a global point of access for a resource.
* To create just one instance of a class, throughout the lifetime of a program.

class language:  
 \_instance=None  
 def \_\_new\_\_(cls):  
 if(cls.\_instance is None):  
 cls.\_instance=super().\_\_new\_\_(cls)  
 return cls.\_instance  
 def print(self):  
 print(**"in print method"**)  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
 obj1=language();  
 obj2=language();  
 if(obj1==obj2):  
 print(**"obj1 and obj2 is equal"**)  
 else:  
 print(**"obj1 and obj2 is not equal"**)

## Monkey Patching

#Monkey patching is a technique to modify or extend the behavior of existing code at runtime.

In Python, the term monkey patch refers to dynamic (or run-time) modifications of a class or module. In Python, we can actually change the behavior of code at run-time.

In Python refers to the practice of dynamically modifying or extending code at runtime typically replacing or adding new functionalities to existing [modules](https://www.tutorialspoint.com/python/python_modules.htm), [classes or methods](https://www.tutorialspoint.com/python/python_classes_objects.htm) without altering their original source code. This technique is often used for quick fixes, debugging or adding temporary features.

class Sample:  
 def show(self):  
 print(**'You are in existing method'**)  
 def modifedShow(self):  
 print(**'You are in enhanced method'**)  
sample = Sample()  
sample.show = sample.modifedShow

# Monkey patching MyClass with modifedShow method  
sample.show()

## Function Decorator

In [Python](https://www.geeksforgeeks.org/python-programming-language-tutorial/), decorators are a powerful and flexible way to modify or extend the behavior of functions or methods, without changing their actual code. A decorator is essentially a [function](https://www.geeksforgeeks.org/python-functions/) that takes another function as an argument and returns a new function with enhanced functionality.  Allowing us to add additional functionality to existing functions or methods in a clean, reusable way.

*# A simple decorator function*def deco(func1):  
 def wrap1():  
 print(**"Before calling the function.Line1"**)  
 print(**"Before calling the function.Line2"**)  
 func1()  
 print(**"After calling the function."**)  
 return wrap1  
*# Applying the decorator to a function*@deco  
def greet():  
 print(**"Hello, World!"**)  
greet()

# Machine Learning

Best Python libraries for Machine Learning

Machine Learning, as the name suggests, is the science of programming a computer by which they are able to learn from different kinds of data. A more general definition given by Arthur Samuel is – “Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.” They are typically used to solve various types of life problems.  
In the older days, people used to perform Machine Learning tasks by manually coding all the algorithms and mathematical and statistical formula. This made the process time consuming, tedious and inefficient. But in the modern days, it is become very much easy and efficient compared to the olden days by various python libraries, frameworks, and modules. Today, Python is one of the most popular programming languages for this task and it has replaced many languages in the industry, one of the reason is its vast collection of libraries. Python libraries that used in Machine Learning are:

* Numpy
* Scipy
* Scikit-learn
* Theano
* TensorFlow
* Keras
* PyTorch
* Pandas
* Matplotlib

# What is PIP?

<https://pip.pypa.io/en/stable/cli/pip_install/>

Package installer of python--PIP

PIP is a package manager for Python packages, or modules. That means it’s a tool that allows you to install and manage additional libraries and dependencies that are not distributed as part of the standard library.

**Package management is so important that pip has been included with the Python installer since versions 3.4** for Python 3 and 2.7.9 for Python 2, and it’s used by many Python projects, which makes it an essential tool for every Pythonista.

The concept of a package manager might be familiar to you if you are coming from other languages. **JavaScript uses npm** for package management, **Ruby uses gem**, and **.NET use NuGet**. In Python, pip has become the standard package manager.

## Check PIP version

Navigate your command line to the location of Python's script directory, and type the following:

C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip --version

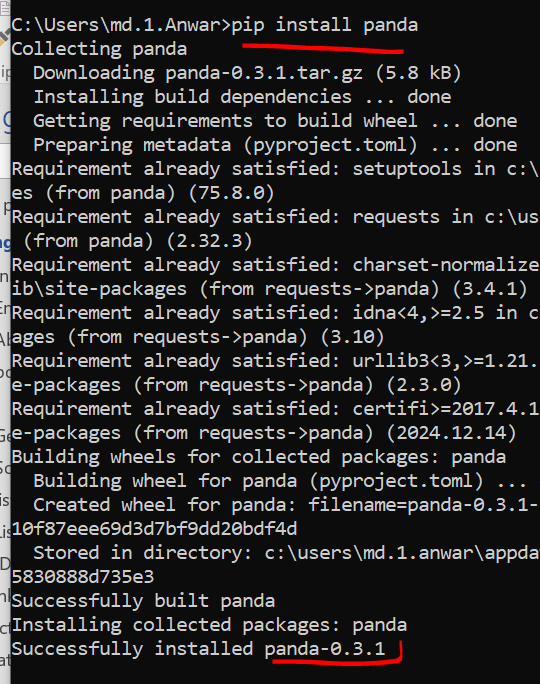
## List Packages

Use the list command to list all the packages installed on your system:

pip list

## Install Module?

pip install <packageName>



Modular programming is a software design technique, which is based on the general principal of modular design. Modular design is an approach, which has been proven as indispensable in engineering even long before the first computers. Modular design means that a complex system is broken down into smaller parts or components, i.e. modules. These components can be independently created and tested. In many cases, they can be even used in other systems as well.

There is hardly any product nowadays, which doesn't heavily rely on modularisation, like cars, mobile phones. Computers belong to those products which are modularised to the utmost. So, what's a must for the hardware is an unavoidable necessity for the software running on the computers.

So far we haven't explained what a Python module is. To put it in a nutshell: every file, which has the file extension .py and consists of proper Python code, can be seen or is a module! There is no special syntax required to make such a file a module. A module can contain arbitrary objects, for example files, classes or attributes. All those objects can be accessed after an import. There are different ways to import a modules.

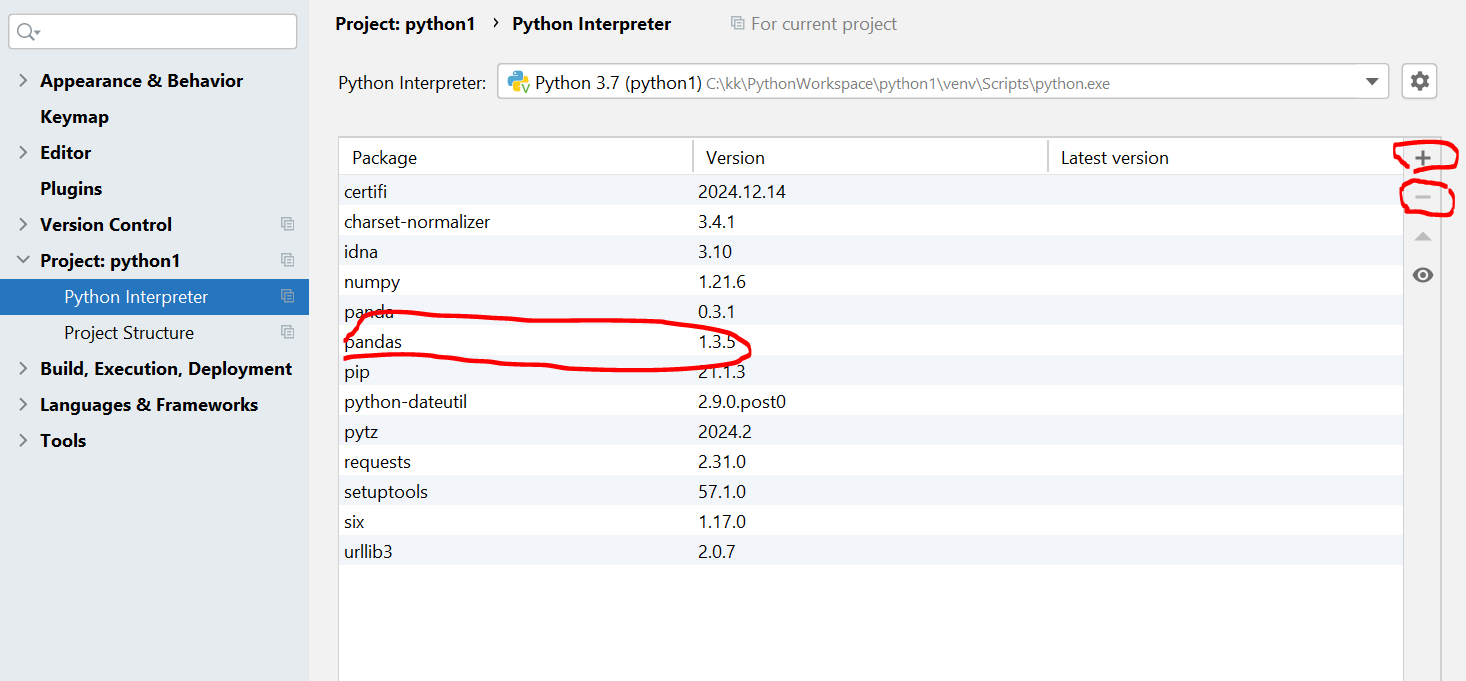
## Install package in PyCharm

File🡪Setting🡪Project>>Python Interpreter

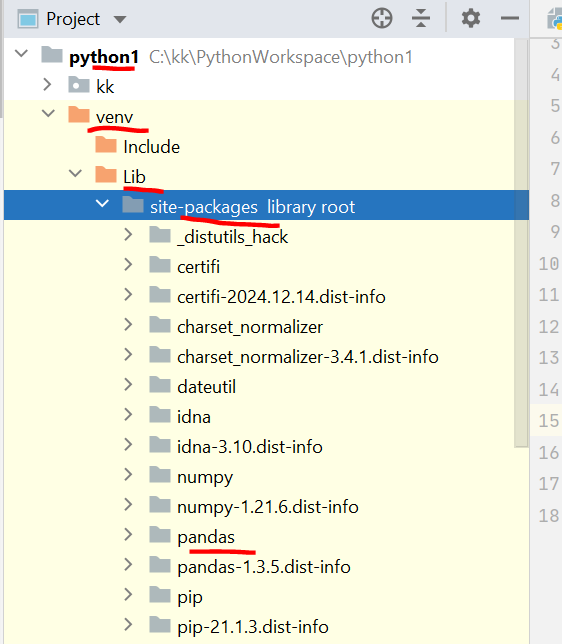
By Default—List of all installed package for a project will be shown

**+** --- Click Plus sign to install

**-** -- Click minus sign to uninstall

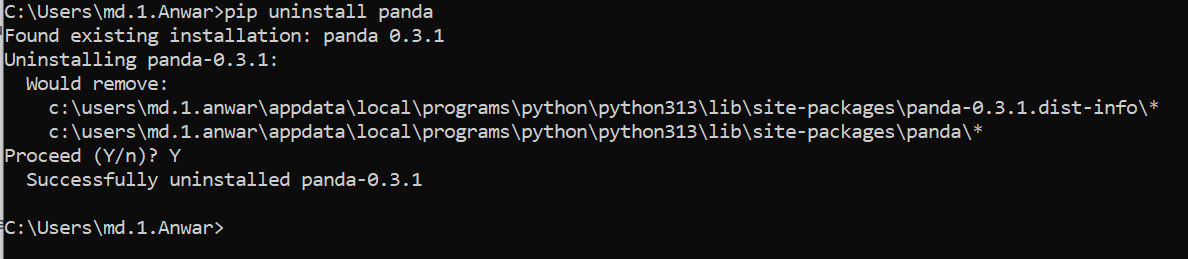


After installation, this package folder must be available in project “**venv”** location as below. Moreover this package name should be list in above screen as well.



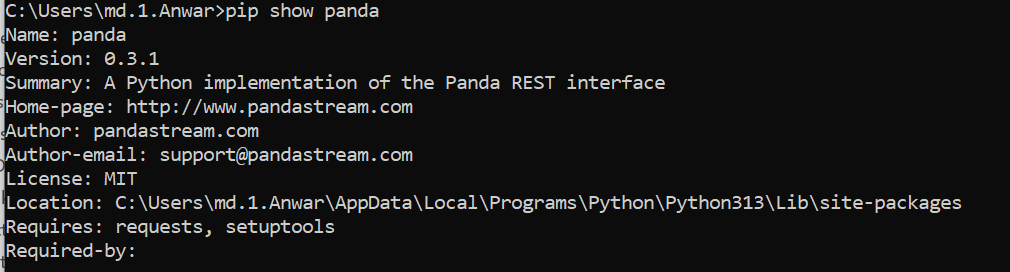
## Uninstall module

Pip uninstall <packageName>



## Show Module details

Pip show <packageName>



# What Is a Virtual Environment?

At its core, the main purpose of Python virtual environments is to create an isolated environment for Python projects. This means that each project can have its own dependencies, regardless of what dependencies every other project has.

The great thing about this is that there are no limits to the number of environments you can have since they’re just directories containing a few scripts. Plus, they’re easily created using the virtualenv or pyenv command line tools.

Below are folder structure of virtual environment.

├── bin

│ ├── activate

│ ├── activate.csh

│ ├── activate.fish

│ ├── easy\_install

│ ├── easy\_install-3.5

│ ├── pip

│ ├── pip3

│ ├── pip3.5

│ ├── python -> python3.5

│ ├── python3 -> python3.5

│ └── python3.5 -> /Library/Frameworks/Python.framework/Versions/3.5/bin/python3.5

├── include

├── lib

│ └── python3.5

│ └── site-packages

└── pyvenv.cfg

Here’s what each folder contains:

* bin: files that interact with the virtual environment
* include: C headers that compile the Python packages
* lib: a **copy of the Python version** along with a **site-packages** folder where each dependency is installed

## ****Why do we need a virtual environment?****

<https://www.geeksforgeeks.org/python-virtual-environment/>

Imagine a scenario where you are working on two web-based Python projects one of them uses [Django](https://www.geeksforgeeks.org/django-tutorial/) 4.0 and the other uses Django 4.1 (check for the latest Django versions and so on). In such situations, we need to create a virtual environment in Python that can be really useful to maintain the dependencies of both projects.

## ****When and where to use a virtual environment?****

By default, every project on your system will use these same directories to store and retrieve site packages (third-party libraries).

This is a real problem for Python since it can’t differentiate between versions in the “site-packages” directory. So both v1.9 and v1.10 would reside in the same directory with the same name.

This is where virtual environments come into play. To solve this problem, we just need to create two separate virtual environments for both projects.

# What is import

<https://chrisyeh96.github.io/2017/08/08/definitive-guide-python-imports.html>

Import in python is similar to #include header file in C/C++. Python modules can get access to code from another module by importing the file/function using import.

* module: any \*.py file. Its name is the **file name**.
* object: in Python, almost everything is an object - functions, classes, variables, etc.
* object: in Python, almost everything is an object - functions, classes, variables, etc.
* When a module is imported, Python runs all of the code in the module file. When a package is imported, Python runs all of the code in the package’s \_\_init\_\_.py file, if such a file exists. All of the objects defined in the module or the package’s \_\_init\_\_.py file are made available to the importer.
* Also, Python imports are **case-sensitive**. import Spam is not the same as import spam.

## Package Versus Module

In Python, a module is a single file that contains code, while a package is a folder that contains multiple modules.

Modules are reusable code that can be imported into other programs, while packages are a way to organize related modules into a single namespace.

**Example**

import math

print(math.pi)

In the above code module math is imported, and its variables can be accessed by considering it to be a class and pi as its object.

It's possible to import more than one module in one import statement. In this case the module names are separated by commas:

import math, random

import statements can be positioned **anywhere in the program**, but it's good style to place them directly at the beginning of a program.

## Syntax

There are 4 different syntaxes for writing import statements.

1. import **<packageName>**
2. import **<moduleName>**
3. from **<packageName>** import <module or subpackage or object>
4. from **<moduleName>** import <object>
5. import **<packageName>** as **<UserDefinedName> --**can be renamed as user defined name

Let X be whatever name comes after import.

* If X is the name of a module or package, then to use objects defined in X, you have to write X.object.
* If X is a variable name, then it can be used directly.
* If X is a function name, then it can be invoked with X()

Optionally, as Y can be added after any import X statement: import X as Y. This renames X to Y within the script. Note that the name X itself is no longer valid. A common example is import numpy as np.

## Using from ... import

pi as whole can be imported into our intial code, rather than importing the whole module.

from math import pi

When you import modules this way, you can refer to the functions by name rather than through dot notation

Instead of using math.pi , we can directly use pi

All the functions and constants can be imported using \*.

from math import \*

## Aliasing Modules

It is possible to modify the names of modules and their functions within Python by using the as keyword.

You may want to change a name because you have already used the same name for something else in your program, another module you have imported also uses that name, or you may want to abbreviate a longer name that you are using a lot.

import [module] as [another\_name]

import math as m

print(m.pi)

print(m.e)

--

* Modules are Python .py files that consist of Python code. Any Python file can be referenced as a module. A Python file called hello.py has the module name of hello that can be imported into other Python files or used on the Python command line interpreter.
* Modules can define functions, classes, and variables that you can reference in other Python .py files or via the Python command line interpreter.
* In Python, modules are accessed by using the import statement. When you do this, you execute the code of the module, keeping the scopes of the definitions so that your current file(s) can make use of these.

import matplotlib

If matplotlib is not installed, you’ll receive an error like this-- ImportError: No module named 'matplotlib'

it can be installed by --pip install matplotlib

import matplotlib.pyplot as plt

plt.plot(x, y)

--OR

import matplotlib as mpl

mpl.pyplot.plot(x, y)

### How to list out all function in a module?

Using the **dir()** Function

import platform

x = dir(platform)

print(x)

### sys.path

To see what is in sys.path, run the following in the interpreter or as a script:

import sys

print(sys.path)

A list of strings that specifies the search path for modules

### How to get a list of built-in modules in python?

import sys

a = sys.builtin\_module\_names

for i in a:

print(i)

## JSON in Python

import json

# some JSON:

x = '{ "name":"John", "age":30, "city":"New York"}'

# parse x:

y = json.loads(x)

# the result is a Python dictionary:

print(y["age"])

# How to use pyCharm tool

## What is Project Interpreter?

### pyCharm Configuration

We can install multiple python version in your PC. In multiple version case, you can choose your version in pyCharm.

In theFile**>>** Settings/Preferences dialog (Ctrl+Alt+S), select Project <project name> | Project Interpreter.

Expand the list of the available interpreters and click the Show All link. Alternatively, click the The Configure project interpreter icon and select Show All.

### How to install Numpy Module

<https://www.datasciencelearner.com/how-to-install-numpy-in-pycharm/>

**Step1**: Go to the File and click on Settings.

**Step** 2: You will see > Project: your\_project\_name.Click on it. You will see two option one is Project Interpreter and other Project Structure.

**Step** 3: Click on the Project Interpreter. You will see all the packages installed.

**Step** 4: You will see the + button. Click on it and search for the numpy in the search field. You will see the numpy package as the left side and its description, version on the right side.

**Step** 5: Selecting numpy click on the Install Package on the left bottom. It will install the packages.

What is Project Interpreter?

## Change pyCharm Theme

File>>setting>>Apearance & Behaviour>>Appearance>> change Them from dropdown

### Rename File name

Select filename>>Right click>>Refactor>>Rename

### How to debug

Toggle breakpoint Ctrl+F8

Resume program F9

Step over F8

Step into F7

Stop Ctrl+F2

# Jupyter Noterbook

## Installation

[Project Jupyter | Installing Jupyter](https://jupyter.org/install)

Pip install JupyterLab #for installing jupyterlab

jupyter lab #launching jupyterlab

# File Handling

**Open**(“FilePath”,”FileMode”) **Read**() **Write**(“String”) **close**()

|  |  |
| --- | --- |
| **Mode** | **Description** |
| ‘r’ | Open text file for reading. Raises an I/O error if the file does not exist. |
| ‘r+’ | Open the file for reading and writing. Raises an I/O error if the file does not exist. |
| ‘w’ | Open the file for writing. Truncates the file if it already exists. **Creates a new file if it does not exist.** |
| ‘w+’ | Open the file for reading and writing. Truncates the file if it already exists. **Creates a new file if it does not exist**. |
| ‘a’ | Open the file for writing. The data being written will be inserted at the end of the file. **Creates a new file if it does not exist.** |
| ‘a+’ | Open the file for reading and writing. The data being written will be inserted at the end of the file. **Creates a new file if it does not exist.** |

example:

file = open('/content/kk.txt', 'w')

file.write('This is the first line.\n')

file.write('This is the second line.')

file.close()

file = open('/content/kk.txt', 'r')

content = file.read()

print(content)

# file.close()

file.read()

file.seek(20) //moving cursor at index 20

print(file.read()) // reading from 20th index

print(file.tell()) //tell current cursor index

file.close()

example—

## How to Create a Text File

<https://www.guru99.com/reading-and-writing-files-in-python.html>

## ****open(filename, mode)****

Open() method return file object.

f= open("guru99.txt","w+")

1. We declared the variable f to open a file named textfile.txt. Open takes 2 arguments, the file that we want to open and a string that represents the kinds of permission or operation we want to do on the file

2. Here we used "w" letter in our argument, which indicates write and the plus sign that means it will create a file if it does not exist in library

3. "w" for write are "r" for read and "a" for append and plus sign means if it is not there then create it

4.One must keep in mind that the mode argument is not mandatory. If not passed, then Python will assume it to be “ r ” by default.

## Write(String)

It writes the contents of string to the file. It has no return value. Due to buffering, the string may not actually show up in the file until the flush() or close() method is called.

for i in range(10):

f.write("This is line %d\r\n" % (i+1))

Step3:

f.close()

## ****writelines(**list**)****

# Writing a file

f = open(\_\_file\_\_, 'a+')

lines = f.readlines() #return multiple line in **list**

f.writelines(lines) -- it requires list as input which will be written in multiple lines

f.close()

**Another example**

List = ["1st line\n", "2nd line\n", "3rd line\n"]

file.writelines(List)

## How to Append Data to a File

You can also append a new text to the already existing file or the new file.

f=open("guru99.txt", "a+")

Step2:

for i in range(10):

f.write("This is line %d\r\n" % (i+1))

Step3:

f.close()

## read()

If file content is small then read() method returns string as full content

f=open("guru99.txt", "r")

if f.mode == 'r':

contents =f.read()

## readlines()

Especially if the file is not too large, it's more convenient to read the file into a complete data structure, e.g. a string or a list. The file can be closed after reading and the work is accomplished on this data structure:

f=open("guru99.txt", "r")

fl =f.readlines() ## returns list [‘1 line**\n**’,’2 line**\n**’]

for x in fl:

print x

## readline():

It reads the **first line of the file** i.e till a newline character or an EOF in case of a file having a single line and returns a string. Like header of file.

## rstrip() / lstrip() Function

Now we want to finally open and read a file. The method rstrip() in the following example is used to strip off whitespaces (newlines included) from the right side of the string "line":

fobj = open("ad\_lesbiam.txt")

for line in fobj:

print(line.rstrip())

fobj.close()

rstrip(): This function strips each line of a file off spaces from the right-hand side.

lstrip(): This function strips each line of a file off spaces from the left-hand side.

## Newlines

A file object that has been opened in universal newline mode have this attribute which reflects the newline convention used in the file. The value for this attribute are “\r”, “\n”, “\r\n”, None or a tuple containing all the newline types seen.

## With open Keyword

You will often find the with statement for reading and writing files. The advantage is that the file will be automatically closed after the indented block after the with has finished execution:

with open("/content/my\_file.txt", "w+") as file:

    # Writing data to the file

    file.write("This is the first line.\n")

    file.write("This is the second line.\n")

    file.seek(0)

    # Read and print the content

    content = file.read()

    print(content)

## encoding

Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings).

Moreover, the default encoding is platform dependent. In windows, it is 'cp1252' but 'utf-8' in Linux.

So, we must not also rely on the default encoding or else our code will behave differently in different platforms.

Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

f = open("test.txt",mode = 'r',encoding = 'utf-8')

## os Module

**os.rmdir()** method in Python is used to remove or delete a empty directory. OSError will be raised if the specified path is not an empty directory.

# Python program to explain os.rmdir() method

# importing os module

import os

# Directory name

directory = "Geeks"

# Parent Directory

parent = "D:/Pycharm projects/"

# Path

path = os.path.join(parent, directory)

# Remove the Directory

# "Geeks"

os.rmdir(path)

**os.remove()** method in Python is used to remove or delete a file path. This method can not remove or delete a directory. If the specified path is a directory then OSError will be raised by the method.

# Python program to explain os.remove() method

# importing os module

import os

# File name

file = 'file1.txt'

# File location

location = "D:/Pycharm projects/GeeksforGeeks/Authors/Nikhil/"

# Path

path = os.path.join(location, file)

# Remove the file

# 'file.txt'

os.remove(path)

# Panda

<https://www.youtube.com/watch?v=NcA8gqv_BGE&list=PLz5n9ljUmyOyPZFolW10TbtZHg9KOis2v> --PlayList

## Series

A Pandas Series is like only **single column** in a table. **Multiple column** is called dataframe

import pandas as pd

a = [1, 7, 2]  
myvar = pd.Series(a)  
print(myvar) #print entire series

print(myvar[0]) #print first value or 0 index value

myvar = pd.Series(a, index = ["x", "y", "z"]) #named index

print(myvar["y"]) #print value at index y (named index)

## DataFrame

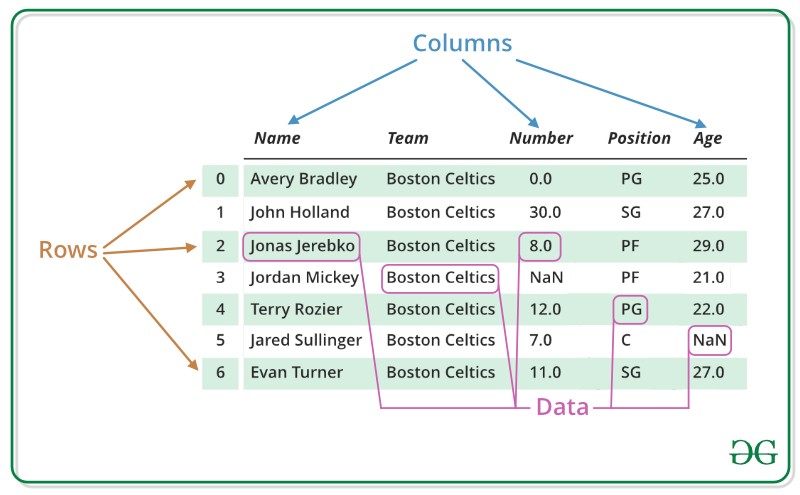
<https://www.geeksforgeeks.org/python-pandas-dataframe/>

Pandas is built on top of the NumPy package, meaning a lot of the structure of NumPy is used or replicated in Pandas. Data in pandas is often used to feed statistical analysis in SciPy, plotting functions from Matplotlib, and machine learning algorithms in Scikit-learn

Pandas DataFrame is two-dimensional size-**mutable**, potentially **heterogeneous tabular data structure** with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

Pandas DataFrame consists of three principal components.

1. **Data**
2. **Rows**
3. **columns**



### Creating a Pandas DataFrame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be

SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc.

#### Dataframe sources

* + **RDBMS**
  + **CSV File**
  + **Excel File**
  + **List**
  + **Dictionary**
  + **List of Dictionary**

#### DataFrame from List

import pandas as pd

lst = ['Geeks', 'For', 'Geeks', 'is','portal', 'for', 'Geeks'] # list of strings

df = pd.DataFrame(lst) # Calling DataFrame constructor on list

print(df)

#### DataFrame from dictionary

To create DataFrame from dict of narray/list, all the narray must be of same length.

import pandas as pd

# intialise data of lists.

#data must be in the form of either array or Key-Array pair

data = {'Name':['Tom', 'nick', 'krish', 'jack'],'Age':[20, 21, 19, 18]}

df = pd.DataFrame(data) # Create DataFrame

print(df) # Print the output.

#### DataFrame with user-defined Index

df1=pd.DataFrame(data, index=['Jan', 'Feb', 'Mar', 'Apr'])

Number of index should be as per number of record in data

print(df1.loc["Feb"]) # locate named index

#### Reading Columns

In Order to select a column in Pandas DataFrame, we can either access the columns by calling them by their columns name.

import pandas as pd

# Define a dictionary containing employee data

data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],

'Age':[27, 24, 22, 32],

'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],

'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

df = pd.DataFrame(data) # Convert the dictionary into DataFrame

print(df**[[**“Name”, 'Qualification'**]]**) # it accept list of columns

To read each value of column

for i in df.index:

print(df['Name'][i])

We can save an entire column into a list:

listVariable = df['Name']

To List column header

df.columns.ravel()

#### Reading Rows

<https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/>

##### **head(n)**

Top n record

by default first five record using df.head(), but we could also pass a number as well:

df.**head**(10) #would output the top ten rows.

##### **tail(n)**

Last n record

To see the last five rows use .tail().

df.tail(2) ##also accepts a number, and in this case we printing the bottom two rows.

##### **info()**

It should be one of the very first commands you run after loading your data:

df.info() ##Tells about all **column name**, **data Type** and dataframe size. Similar to **desc table**

##### Shape

It is variable name which returns (**NoOfrows**, **NoOfcolumns**) of dataframe

Another fast and useful attribute is .shape, which outputs just a tuple of (rows, columns)

df.shape ## returns (**NoRows**, **NoCols**)

df.columns ##return list of all column name

##### .describe()

Using describe() on an entire DataFrame we can get a summary of the distribution of continuous variables:

df.describe()

df['genre'].describe()

##### .to\_string(index=False)

print(df.to\_string(index=False)) #return dataframe format without index

df.iloc[[rowidex],[colIndex]]

This is an integer locator

Df.iloc[[0,1]] #return entire column for row index 0 and 1

Df.iloc[[0,1], [0,1]] #return 0 and 1 index column for row index 0 and 1

df.loc[[rowidex],[colNameList]]

df.loc[[0,1],[‘col1’,’coln’]] #return col1 and coln column for row index 0 and 1

df.set\_index(“colName”)

In this case , mentioned column name will be index of dataframe. We can search column value like below

Df.loc[[“value1”,”Value2”]] #value1 and value2 are part of index column value

#### Sorting

##### sort\_values()

**Syntax**

dataframe.sort\_values(by, axis, ascending, inplace, kind, na\_position, ignore\_index, key)

**Return Value**

**DataFrame** with the sorted result,

or

**None** if the inplace parameter is set to True.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Value** | **Description** |
| by | String | Required. Specify labels to sort by. Either index level(s) or column label(s). Or, if the axis is 1 or 'columns' this values specify column level(s) or index label(s) |
| List of strings |
| axis | 0 | Optional. Default 0. Specifies the axis to sort by |
| 1 |
| 'index' |
| 'columns' |
| ascending | True | Optional, default True. Specifies whether to sort ascending (0 -> 9) or descending (9 -> 0) |
| False |
| inplace | True | Optional, default False. Specifies whether to perform the operation on the original DataFrame or not, if not, which is default, this method returns a new DataFrame |
| False |
| kind | 'quicksort' | Optional, default 'quicksort'. Specifies the sorting algorithm |
| 'mergesort' |
| 'heapsort' |
| na\_position | 'first' | Optional, default 'last'. Specifies how to handle NULL values. 'first' means put them first, 'last' means put them last. |
| 'last' |
| ignore\_index | True | Optional, default False. Specifies whether to ignore index or not. If True the original indexes are ignored, and replaced by 0, 1, 2 etc. |
| False |
| key | Function | Optional, specify a function to be executed before the sorting |

Pandas sort\_values() function sorts a data frame in Ascending or Descending order of passed Column.

Example #1: Sorting by column **Name** and **Team**

In the following example, A data frame is made from the csv file and the data frame is sorted in ascending order of Team and in every Team the Name is also sorted in Ascending order.

import pandas as pd

data=pd.read\_csv("nba.csv") ##making data frame from csv file

data.sort\_values(["Team", "Name"], axis=0, ascending=True, inplace=True)

print(data) #display

##### sort\_index()

Sorting by the labels of the DataFrame

By contrast, sort\_index doesn’t indicate its meaning as obviously from its name alone. The key thing to know is that the Pandas DataFrame lets you indicate which column acts as the row index. And if you didn’t indicate a specific column to be the row index, Pandas will create a zero-based row index by default.

data = {  
  "age": [50, 40, 30, 40, 20, 10, 30],  
  "qualified": [True, False, False, False, False, True, True]  
}  
idx = ["Mary", "Sally", "Emil", "Tobias", "Linus", "John", "Peter"]  
df = pd.DataFrame(data, index = idx)  
newdf = df.sort\_index()

Therefore, when you execute sort\_index, you’re sorting the DataFrame by its row index.

#### Merging

##### Inner Merge / Inner join

The default Pandas behaviour, only keep rows where the merge “on” value exists in both the left and right dataframes.

##### Left Merge / Left outer join

(aka left merge or left join) Keep every row in the left dataframe. Where there are missing values of the “on” variable in the right dataframe, add empty / NaN values in the result.

##### Right Merge / Right outer join

(aka right merge or right join) Keep every row in the right dataframe. Where there are missing values of the “on” variable in the left column, add empty / NaN values in the result.

##### Outer Merge / Full outer join

A full outer join returns all the rows from the left dataframe, all the rows from the right dataframe, and matches up rows where possible, with NaNs elsewhere.



##### merge indicator

To assist with the identification of where rows originate from, Pandas provides an “indicator” parameter that can be used with the merge function which creates an additional column called “\_merge” in the output that labels the original source for each row.

result = pd.merge(df1,df2,on='use\_id',how='outer',indicator=True)

Using left\_on and right\_on to merge with different column names

#### Filtering

Let us say we want to filter the data frame such that we get a smaller data frame with “year” values equal to 2002. That is, we want to subset the data frame based on values of year column. We keep the rows if its year value is 2002, otherwise we don’t.

print(df[df["year"]=="2002"])

Or

print(df[df.year=="2002"])

print(df[df["year"]!="2002"])

##### isin method

Pandas dataframe’s **isin()** function allows us to select rows using a list or any iterable.

If we use isin() with a single column, it will simply result in a boolean variable with True if the value matches and False if it does not.

**Syntax**

dataframe.isin(values) #values could be list,Tuple *,*series, dataframe, disctionary

df.isin([1,2,3]) # input as list

df.isin(df[‘col1’) # input as panda series

Example1

# creating filters of **bool** series from isin() –bool datatype

filter1 = data["Gender"].isin(["Female"])

filter2 = data["Team"].isin(["Engineering", "Distribution", "Finance" ])

# displaying data with both filter applied and mandatory

data[filter1 **&** filter2] # and condition

data[filter1 **|** filter2] #OR condition

Example

df1=pd.read\_sql(**"SELECT \* FROM stringinteger"**, mydb)  
df2=pd.read\_sql(**"SELECT \* FROM stringinteger WHERE id<=20"**, mydb)  
print(df1[~df1[**'id'**].isin(df2[**'id'**])]) #df1 minus df2

print(df2[~df2[**'id'**].isin(df1[**'id'**])]) #df2 minus df1

##### isin method -Negation

We can also select rows based on values of a column that are not in a list or any iterable. We will create boolean variable just like before, but now we will negate the boolean variable by placing **~** in the front.

continents = ['Asia','Africa', 'Americas', 'Europe']

df2 = df[~df[‘continent’].isin(continents)]

This will result in a smaller dataframe with gapminder data for just Oceania continent.

##### And/OR Conditions

We can combine multiple conditions using & operator to select rows from a pandas data frame.

df2=df[df[‘continent’].isin(continents) **&** df[‘year’].isin(years)]

df2=df[df[‘continent’].isin(continents) **|** df[‘year’].isin(years)]

##### nonull()

Year column which does not 'NaN'

df[df.year.notnull()]

Year column which have 'NaN'

df[~df.year.notnull()]

##### And(&) OR(|)

print(df[~df[**"Name1"**].notnull() | ~df[**"Name2"**].notnull()])

print(df[~df[**"Name1"**].notnull() & ~df[**"Name2"**].notnull()])

##### .query()

df[df.country=="United States"]

OR

df.query('country=="United States"')

df.query('year==1952')

df.query('country=="United States" & year > 1996')

df.query('country in ["United States", "United Kingdom"] & year > 2000')

#### Summarising, Aggregating

<https://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/>

#### .count()

#### .max()

#### .sum()

#### .value\_counts()

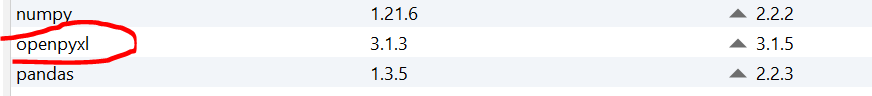
#### .groupby()

<https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/>

#### .agg()

### Read Excel Sheet

Dependent module “**openpyxl**” is required to be installed to run **read\_excel()** function



df=pd.read\_excel(“xlsxFileName”)

df = pd.read\_excel('records.xlsx', sheet\_name='Cars', usecols=['Car Name', 'Car Price'])

### Reading CSV File

**import** pandas **as** pd  
f1 = pd.**read\_csv**(**'file1.txt'**, sep=**','**)

### Reading Pipe delimited File

### f2 = pd.read\_csv('file2.txt',sep='|') Reading tab delimited File

f2 = pd.**read\_csv**(**'file2.txt'**,sep=**'\t'**)

f2 = pd.**read\_table**(**'file2.txt'**,sep=**'\t'**) #special function to tab delimited file

pandas.isin - Return boolean DataFrame showing whether each element in the DataFrame is contained in values.

python tilde ~ - Bitwise not, inversing boolean - False to True and True to False

### Reading JSON File

Big data sets are often stored, or extracted as JSON.

**json**=**python dictionary**

df = pd.read\_json(**'data.json'**)  
print(df.to\_string())

### writing to csv file

df.**to\_csv**(“file1.csv”)) # Dataframe will be written in file1.csv file in comma separated delimitor

but index will also be written. If you don’t want to write index then below code

df.**to\_csv**(**"../Files/data1.csv"**,index=False)

df.to\_dict(orient='record'))

df.to\_json(orient='records'))

### Deal with file without header

Pd.read\_csv(“filename”,header=None) #be default 0,1,2,.. will be coming in place of header name

If you want to provide user defined column name then as below

Colnames=[‘col1’,’col2’,’col3’] ##list of column name and same has to be pass with parameter **names** as below

Pd.read\_csv(“filename”,**header**=None,**names**=colnames)

## Panda Merge

<https://www.tutorialspoint.com/python_pandas/python_pandas_merging_joining.htm>

Panda Concatenation

## Excel Sheet with Panda

<https://www.dataquest.io/blog/excel-and-pandas/>

excel\_file = 'movies.xls'

movies = pd.read\_excel(excel\_file)

Here, the read\_excel method read the data from the Excel file into a pandas DataFrame object. Pandas defaults to storing data in DataFrames. We then stored this DataFrame into a variable called movies.

Pandas has a built-in DataFrame.head() method that we can use to easily display the first few rows of our DataFrame. If no argument is passed, it will display first five rows. If a number is passed, it will display the equal number of rows from the top.

Excel files quite often have multiple sheets and the ability to read a specific sheet or all of them is very important. To make this easy, the pandas read\_excel method takes an argument called sheetname that tells pandas which sheet to read in the data from. For this, you can either use the sheet name or the sheet number. Sheet numbers start with zero. If the sheetname argument is not given, it defaults to zero and pandas will import the first sheet.

By default, pandas will automatically assign a numeric index or row label starting with zero. You may want to leave the default index as such if your data doesn’t have a column with unique values that can serve as a better index. In case there is a column that you feel would serve as a better index, you can override the default behaviour by setting index\_col property to a column. It takes a numeric value for setting a single column as index or a list of numeric values for creating a multi-index.

## Write to excel

<https://pythonexamples.org/pandas-write-dataframe-to-excel-sheet/>

### openpyxl Module

pip install openpyxl

You can write the DataFrame to Excel File without mentioning any sheet name. The step by step process is:

Have your DataFrame ready.

Create an Excel Writer with the name of excel file you would like to write to.

Call to\_excel() function on the DataFrame with the writer passed as argument.

Save the Excel file using save() method of Excel Writer.

# render dataframe as html

writer = pd.ExcelWriter('output.xlsx')

df\_marks.to\_excel(writer)

writer.save()

print('DataFrame is written successfully to Excel File.')

### Write DataFrame to a specific Excel Sheet

# render dataframe as html

writer = pd.ExcelWriter('output.xlsx')

df\_marks.to\_excel(writer, 'marks')

writer.save()

print('DataFrame is written successfully to Excel Sheet.')

# ETL Testing with panda

## Find column list

print(df.columns) # list of column  
print(df.count()) #column and their no of record count

## Find unique value of column

df=pf.read\_csv(“csvFileName”)

print(df[**"name1"**].unique())

## Find duplicate rows in a file

df=pf.read\_csv(“csvFileName”)

print(df[df.duplicated()]) #returns all only duplicate record

if you want to know duplicate in any specific column then below code

print(df[df[**'col1'**].duplicated()])

print(df.duplicated().sum()) # return no of duplicate record(numbers only)

df[df.duplicated()].to\_csv(“csvFile”) # to write duplicate record in csv file

## Find top 2 salary

df=pf.read\_csv(“csvFileName”)

print(df[**'salary'**].**nlargest**(2)) #top 2 salary in a column

## Find lowest 2 salary

df=pf.read\_csv(“csvFileName”)

print(df[**'salary'**].**nsmallest**(2))

## Apply filter on column

filt=df[**'salary'**]>3000  
print(df[filt])

filt1=df['salary']!=3000

print(df[filt1])

## Group BY

g=df.groupby(**'col1'**)  
print(g[**'col1'**].count())

print(g.get\_group(**'A1'**)) #based on grouped value, we can filter by value.

## Null record count

print(df[df[**"name2"**].isnull()]) #returns column name2 null record

## Dataframe method list

<https://www.programiz.com/python-programming/pandas/methods/dataframe>

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>

# MySql Connector

## Module Name

“mysql-connector-python”

### Python Code

import mysql.connector  
mydb = mysql.connector.connect(  
 host=**"localhost"**,  
 user=**"root"**,  
 password=**"root"**,  
 database=**"etl"**)  
print(mydb)  
mycursor = mydb.cursor()  
mycursor.execute(**"INSERT INTO stringinteger values(24,'Hello Python')"**)  
mydb.commit(); *#commit is required for DML statement* ***update****/****insert****/****delete***mycursor.execute(**"SELECT \* FROM stringinteger"**)   
print(**"success"**)  
for x in mycursor: *#for reading all returned record* print(x)

fetchone() – Fetch top 1 record

myresult = mycursor.fetchone()

fetchall() –Fetch all record

myresult = mycursor.fetchall()

## Module Name for pandas

1. PyMySQL ---Dependent module
2. SQLAlchemy --Main Module

### Python Code

from sqlalchemy import create\_engine  
import pandas as pd  
connStr = **'mysql+pymysql://root:root@localhost:3306/etl'**engine = create\_engine(connStr)  
MySqlConn=engine.connect();  
print(MySqlConn)  
df=pd.read\_sql(**"SELECT \* FROM stringinteger"**, MySqlConn)  
print(df)