Features of Python :-
High Level Language
Free & Open Source

Installation :
Python.org

Ex: print("Hello World") Save it hello.py
to Run: python hello.py ↵

Pip : is package manager for python . to install module we use pip
ex: pip install Pandas (it will install Pandas in

Types of Modules

Built-in Modules
- Pre-installed in python
ex: os, abc

External Modules
- Not
- ex: Pandas, Te flask etc.

Using Python as a Calculator

Comments:
Single line
#

Multiple line
""" ___ """

Q" Use REPL & Print the table of 5 using

Q" Write a python program to print the cur of a directory using OS Module.

cwd: cur working

```
import os
cwd = os.getcwd()
Print("Current working directory:", cwd)
```

o/p:

Note: The CWD is the folder in which Python Script operat

To change Current Working Directory (CWD),
os.chdir() method is used

To Creating a Directory
os.mkdir() or os.makedirs()

his method raise "File Exists Error" - used to create di
the dir to be created already recursively that me
exists while making leaf dir
any intermediate-level dir
missing, os.makedirs() meth
create them all

Listing out Files and Directories with Python:
os.listdir()

Deleting Directory or Files using Python:
os.remove()
os.rmdir()

```
import os
directory = "Geeks"
Parent = "D:/Pycharm Projects/"
Path = os.path.join(parent, direc
os.rmdir(path)
```

To get the name of OS
os.name()

```
import os
print(os.name)
```

O/P: nt (for

s.error

.popen() Open as pipe to or from Command. The
value can be read or written depending o
depending on whether mode is 'r' or 'w'

os.rename() A file old.txt can be renamed to new.txt

```
import os
fd = "GFG.txt"
os.rename(fd, 'New.txt')
os.rename(fd, 'New.txt')
```

O/P: Error

Because file name "GFG.txt" exists, thus when os.rename() is used the first, the files get renamed. Upon calling the os.rename() second time, file "New.txt" exists not "GFG.t

Variables & Data Types:

Variable: Container to store a value

```
a = 30
b = "Preeti"
c = 71.22
```

Keywords: Reserved words in Python

Identifiers: class / fun / Variable name

Data Types: Strings, List, Tuples, Sets, Dictionary, Arra

Integers 2) Floating Point Numbers 3) Strings 4) Booleans 5) N

```
a = 71          (int)
b = 03.44       (floating)
name = "Harry"  (String)
```

Operators in Python :-

Arithmetic operators (+, -, *, /, etc)

Assignment Operators (=, +=, -=, etc)

Comparison operators (==, >=, <=, >, <, != etc)

Logical operators (and or not)

Type() fun" & Typecasting : used to difine data ty of fun"

```
a = 31
type (a)
```

o/p: class <int>

A number can be converted into a string & vice-versa

Str(31) o/p: "31"

int("32") o/p: 32

float(32) o/p: 32.0

Input() fun"

```
a = input("Enter name")
```

Note: The o/p of input fun" is always a string even if the number is entered by the user

Ques" WAP to add two numbers.

```
num1 = 1.5
num2 = 6.3
Sum = num1 + num2
Print ('The sum of {0} and {1} is {2}', format(num1, num2,
```

OR

```
num1 = input ('Enter first number :')
num2 = input ('Enter second number :')
Sum = float(num1) + float(num2)
Print ('The sum of {0} and {1} is {2}'. format(num1, nu
```

Chapter 3 String

```
a = 'harry'      # Single Quoted
b = "harry"      # Double "
c = """harry"""  # Triple "
```

String Slicing : A string in Python can be sliced for getting a part of the string.

name = "Harry" → length = 5

H	a	r	r	y
0	1	2	3	4
(-5)	(-4)	(-3)	(-2)	(-1)

The index in a string starts from 0 to (length - 1)

To slice a String

sl = name[ind_start : ind_end]

first indexed included ↙ ↘ last index is not included

sl [0:3] returns "Har" → 0 to 3

sl [1:3] returns "ar" → 1 to 3

Negative Indices : Same as -1 corresponds to (length - 1)
-2 -2)

Slicing with Skip Value

```
word = "amazing"
word [1:6:2]      o/p : mzn
```

Other advanced Slicing technique

```
word = "amazing"
```

word [:7] or word [0:7] o/p 'amazing'

word [0:] or word [0:7] o/p 'amazing'

String fun":

len() function : return length of the String

ex: len("amazon") O/P 6

endswith("ing") :

count("c") : count total no. of occurences of any character

Capitalize() : Capitalize first character of string

find(word) : find a word & returns the index of first occurence of that word in string.

replace(oldword, newword) : replace old word with new word

Escape Sequence Characters:

ex: \n , \t (tab) , \' (single quote), \\ (backslash) , etc.

3) WAP to detect double spaces in a string

Method 1: Using regex

```
import re
test_str = "My name is Preeti"
print("The origin string is:" + test_str)
res = bool(re.search(r"\s", test_str))
print("Does string contain spaces ?" + str(res))
```

Method 2: Using in operator

```
res = " " in test_str
```

Loops / Conditionals

The for loop in Python is an iterating fun.

Syntax:

```
for iterator_var in sequence
    statements
```

Ex 1)
```
n = 4
for i in range(0, n):
    print(i)
```
O/P:
0
1
2
3

2)
```
word = "anaconda"
for letter in word
    print(letter)
```
O/P:
a
n
a
c
o
n
d
a

Using the loop to iterate over a Python list or tuple

```
words = ["Apple", "Banana", "Car", "Dolphin"]
for word in words:
    print(word)
```

O/P:
Apple
Banana
Car
Dolphin

Nested loop: When we have a for loop inside another for loop, its called Nested for loop

```
words = ["Apple", "Banana", "Car", "Dolphin"]
for word in words
        # This loop is fetching word from the list
    print("The following lines will print each letter of "+word")
    for letter in word:
            # This loop is fetching letter for the word
        print(letter)
    print(" ") # This print is used to print a blank line
```

o/p: The following lines will print each letter of Apple

```
A
p
p
l
e
```

Loop can be used to iterate over a range and iterate

```
n=4
for i in range(0, n):
    print(i)
```

o/p: 0 1 2 3

Iteration Over list

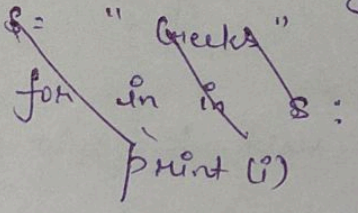
```
print("List iteration")
l = ["geeks", "for", "geeks"]
    for i in l:
        print(i)
```

Iterating over Tuple (immutable)

```
print ("\n Tuple Iteration")
t = ("geeks", "for", "geeks")
for i in t:
    print (i)
```

Iterating Over Dictionary

```
print ("\n Dictionary Iteration")
d = dict()
d['xyz'] = 123
d['abc'] = 345
for i in d
print ("%s %d" % (i, d[i]))
```



O/P: Dictionary Iteration
xyz 123
abc 345

Iterating Over a String

```
print ("\n String Iteration")
s = "Geeks"
for i in s :
    print (i)
```

O/P:
G
e
e
k
s

Iterating by Index of Sequences: The key idea is to first calculate the length of the list & in iterate over the sequence within the range of this length.

?
```
list = ["geeks", "for", "geeks"]
for index in range(len(list)):
    print list[index]
```
O/P: geeks for geeks

Using Else Statement with for loops

```
list = ["geeks", "for", "geeks"]
for index in range(len(list)):
    print list[index]
else:
    print "Inside Else Block"
```
O/P: geeks for geeks Inside Else Block

Nested loops:

```
from __future__ import print_function
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

O/P:
```
1
2 2
3 3 3
4 4 4 4
```

Loop Control Statements : change execution from its normal sequence.

Continue Statement : It returns the control to the begining of the loop.

```
for letter in 'geeksforgeek':
    if letter == 'e' or letter == 's':
        continue
    print 'current letter :', letter
    var = 10
```

why var = 10 ?

O/P: current letter : g
" " k
" " f
" " o
" " r
" " g
" " k

Break Statement : It brings control out of loop

```
for letter in 'geeksforgeeks':
    # break the loop as soon it sees 'e'
    # or 's'
    if letter == 'e' or letter == 's':
        break
    print 'Current letter :', letter
```

I/P: Current letter : e

Pass Statement : use to write empty loops. Pass is also used for empty control Statement, function and classes.

```
# an empty loop
for letter in 'geeksforgeeks':
    pass
print 'Last letter :', letter
```

O/P: Last letter : s

While Loop : while loop is used to execute a block of statements repeatedly until a given a condition satisfied.

```
while expression :
    Statement(s)
```

```
count = 0
while ( count <3):
    count = count+1
    print ("Hello Geek")
```

O/P: Hello Geek
Hello Geek
Hello Geek

else Statement with while loops

```
if condition ;
    # execute these statements
else
    # execute these statements
```

```
while condition
    #
else
    #
```

```
count = 0
while (count <3):
    count = count +1
    print ("Hello Geek")
else:
    print ("In else Block")
```

O/P: Hello Geek
Hello Geek
Hello Geek
In Else Block

- Single Statement while block!

```
count = 0
while (count == 0): print ("Hello Geek")
```

Note: Not suggested to use as its is a never ending infinite loop where the condition is always true & you have to forcefully terminate the Compiler

Looping Techiques in Python: used where we don't need to actually manipulate the structure and order of the overall containers.

enumerate() used to loop through the Containes printing the index number along with the value present in that particular index.

```
for key, value in enumerate (['The', 'Big', 'Bang', 'Theory'])
    print (key, value)
```

O/P:
```
0 The
1 Big
2 Bang
3 Theory
```

```
for key, value in enumerate (['Geeks', 'for', 'Geeks', 'is'
                              'the', 'Best', 'coding', 'Platform'])
    print (value, end = ' ')
```

P: Geeks for Geeks is the Best Coding platform

zip() used to combine 2 similar Containers (list-list or dict-dict) printing the values sequentially.

ex: questions = ['name', 'colour', 'shape']
answers = ['apple', 'red', 'a circle']
# using zip() to combine containers
# and print values
for question, answer in zip(questions, answers):
print('\nWhat is your {0}? I am {1}.'.format(question, answ

o/p: What is your name? I am apple
What is your colour? I am red
What is your shape? I am circle

iteritems() used to loop through the dictionary prin the dictionary key value pairs. sequentially

items() Similar to iteritems but takes more m very time consuming

sorted() to print the container in Sorted Order

Set() to Remove Duplicate

reverse() used to print the values of containe in reversed order

If- else/ Elif

```
a = 3
b = 9
if b % a == 0:
    print " b is divisible by a"
elif b+1 == 10:
    print " Increment in b produces 10"
else:
    print " You are in else Statement"
```

Lists

Dynamic Sized Arrays, mutable (can be altered even after their creation)

Note: Lists are useful tool for preserving a seq. of data & further iterating over it.

```
List = [ ]
print (" Blank List: ")
print (list)
```

```
List = ["Hello", "How"]
Print (List)
```

List Methods:

Append()	Add an element to the end of the list
Extend()	Add all element of a list to another list
Insert()	Insert an item at defined index
Remove()	Remove an item from the list
Pop()	Remove and returns an element at given index
Clear()	Remove all items from the list
Index()	Returns the index of the first matched item
Count()	Returns the count of number of items passed as an argument
Sort()	Sort items in a list in ascending Order
Reverse()	Reverse the order of items in the list
Copy()	Returns Copy of the list

Built - in functions with List

reduce()

Sum() Sums up the numbers in the list

Ord() Returns an integer representing the Unicode point of the given Unicode character

cmp()

max() returns max. element of given list

min()

all()

len()

~~fm~~

enumerate()

accumulate()

filter() tests if each element of a list true or not

map()

lambda()

Negative Indexing: Start with -1 from last

ex: list = [1, 6, 9, "Hello", "geeks"] O/P: Geeks

print (list[-1])

print (list[-3]) O/P : 9

Remove() : to remove item in list

list = [1, 2, 3, "Hi"] O/P: [2, 3, 'Hi']

list.remove(1)

for i in in range (1, 5)
list.remove (i)
print (list)

Pop() : to remove item from end by default
remove item using index.

List = [1, 2, 3, "Hi"]
list.pop(1)
Print (list) → with the help of this return list after poping item

o/p : ~~[1, 3, "Hi"]~~
& [1, 3, "Hi"]

List = [1, 2, 3, "Hi"]
list.pop(2)
o/p: 3

: Slicing List : [Imp]

Slice is operation not method

List = ['G', 'E', 'E', 'K', 'S', 'F', 'O', 'R']
Sliced_list = list[3:6]
Print (Sliced_list)

o/p: ['K' 'S', 'F']

* Here 3 index include in list but index 6 not

Sliced list from begining to end
Sliced_list = list[:]
Print (Sliced_list)

Sliced_list = list[5:] # Sliced from 5th to end

Negative Index Slicing :-

```
Sliced_list = list[:-6]   # Sliced till 6th element from last:")
Sliced_list = list[-6:-1] # Sliced from index -6 to -1
Sliced_list = list[::-1]  # to print list in Reverse
```

Operations on List

- Find length of List
- Iterate over a list in Python [Turn to Page on loop]
- Concatenating two lists in Python
- List Membership Test

Ways to Concatenate two lists in Python:

Method 1: Using Naive Method

```
test_list1 = [1, 4, 5, 6, 5]
test_list2 = [3, 5, 7, 2, 5]

for i in test_list2 :
    test_list.append(i)
```

] Naive Method to Concat

```
print("Concatenated list using Naive Method:" + str(test_list2)
```

Method 2: Using + Operator

```
test_list3 = [1, 4, 5, 6, 5]
test_list4 = [3, 5, 7, 2, 5]
test_list3 = test_list3 + test_list4
print("Concatenated lis using + :" + str(test_list3)
```

2) 'not in' operator

```
x = 24
y = 20
list = [10, 20, 30, 40, 50];
if (x not in list):
    print ("x is NOT present in given list")
else:
    print ("x is present in given list")
if (y in list):
    print ("y is present in given list")
else:
    print ("y is NOT present in given list")
```

Identity operator : used to determine the type of data a certain variable contains

1) 'Is' operator

```
x = 5
if (type(x) is int):
    print ("true")
else:
    print ("false")
```

O/P: True

2) 'Is not' operator

replace is → is not

- Python list comprehension
- Nested list Compr
- list comprehn and ord()

Method 3: Using '*' Operator

```
test_list1 = [1, 4, 5, 6, 5]
test_list2 = [3, 5, 7, 2, 5]
res_list = [*test_list1, *test_list2]
print ("Concatenated list using * operator : " + str (res_list))
```

Python Membership and Identity operators

Membership operators are used to validate the membership of value. It test for membership in seq., such as strings, list, or tuples

'in' operator: used to check if a value exists in a sequence or not.

```
list1 = [1, 2, 3, 4, 5]
list2 = [6, 7, 8, 9]
for item in list1:
    if item in list2:
        print ("Overlapping")
    else:
        print ("not Overlapping")
```

'not in' operator:

set1 = [(3, 4, 5)] ✗

set1 = set(3, 4, 5) ✗

set1 = set([3, 4, 5]) ✓

Sets

Set is an unordered collection of data type that is iterable, mutable

The major advantage of using a set, as opposed to a list, is that it has highly optimized method for checking whether a specific element is contained in the set

Creating Set : created by using built-in set() fun^n

```
Set1 = set()
Print (Set1)
```

```
Set1 = set("geeks")
Print (Set1)
```

Creating a Set with the use of Constructor [Imp]

```
~~Set1 = set("Geeksfor Geeks")~~
String = 'Geeksforgeeks'        # using object to store String
Set1 = set(String)
print (Set1)
```

Create with use of list

```
Set1 = set(["geeks", "for", "geeks"])
Print (Set1)
```

Removing elements from the set

remove()

discard() : remove elements from a set without key error, if the element doesn't exist in the set

```
Set1 = set([1,2,3,4,5,6,7,8,9,10,11,12])
Print(Set1)
Set1.remove(5)
Set1.remove(6)
Print(Set1)
```

```
Set1.discard(7)
Set1.discard(8)
Print(Set1)
```

```
for in in range(1,5):
    Set1.remove(i)
    Print(Set1)
```

Pop() : removes only last element of the list

```
set1.pop()
Print(Set1)
```

clear() : remove all the elements from the set

```
Set1.clear()
Print(Set1)
```

Frozen sets :

Adding Elements to a Set

add() : add only one element at a time, for Multiple use loops

```
Set1 = set()
Set1.add(8)
Set1.add(9)
Print(set1)
```

```
Set1 = set()
for i in range(0,10):
    Set1.add(i)
Print(Set1)
```

list can't be added as set elements

update() to add more than one or multiple elements

```
Set1 = set([4, 5, (6,7)])
Set1.update([10,11])
Print(set1)
```

accepts lists, string, tuples

Accessing a Set :

can't be accessed by referring to an index since sets are unordered

difference ( ) : SetA - SetB

SetA = {10, 20, 30, 40, 80}
SetB = {50, 100, 80, 40}
Print (SetA.difference (SetB))
Print (SetB.difference (SetA))

Or Print (SetA - SetB

Union ( ) : SetA ∪ SetB

intersection ( ) : SetA ∩ SetB

isdisjoint ( ) : SetA ∩ SetB = ϕ (Null)

issubset ( ) :

issuperset ( ) :

Symmetric difference ( )

Creating a tuple with Nested Loop tuples

```
Tuple1 = (0, 1, 2, 3)
Tuple2 = ('python', 'geek')
Tuple3 = (Tuple1, Tuple2)
Print(Tuple3)
```

Creating a tuple with repetition

```
Tuple1 = ('Creeks',) * 3
print(Tuple1)
```

Creating a tuple with loop

```
tuple1 = ('geeks')
n=5
for i in range(int(n)):
    tuple1 = (Tuple1,)
    print(tuple1)
```

Accessing of tuples :-

Tuples

Creating of tuple without the use of parentheses is known as Tuple Packing.

Immutable (ie can't be Modified)

Creating a Tuple

```
Tuple1 = ('Geek', 'For')
Print (tuple1)
```

Creating tuple with the use of list

```
list1 = [1, 2, 3, 4]
print (tuple (list1))
```

Creating tuple with the use of built in fun"

```
tuple1 = tuple ('Geeks')
print (Tuple1)
```

Creating with Mixed Data Types

```
tuple1 = (5, 'Welcom', 7)
Print (tuple1)
```

Concatenation of tuples

```
Tuple1 = (1, 2, 3)
Tuple2 = ('geeks', 'for')
tuple3 = tuple1 + tuple2
Print (tuple1)
Print (tuple2)
Print (tuple3)
```

Slicing of Tuples :

```
tuple1 = tuple ('geeksforgeeks')
print (tuple1[1:])
print (tuple1[::-1])
print (tuple1[4:9])
```

Remove of first element

Reverse the Seq. of Tuple

Print b/w range 4 to 9

Deleting a Tuple

```
tuple1 = (0, 1, 2, 3)
del Tuple1
Print (tuple1)
```

o/p: Error because Tuple is deleted

Built in Methods

index()

count()

Built in fun's

① WAP to find size of tuple

getsizeof() belongs to "sys" module or

-sizeof_() # inbuilt fun"

ex: tuple1._sizeof_()

② Max & Min K elements

8/05/21

Dictionary

Dictionary holds key: value pair.

Note: Keys in dictionary doesn't allows Polymorphism

Dictionary keys are case sensitive, same name but diff cases & key will be treated distinctly

Ex: Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print(Dict)

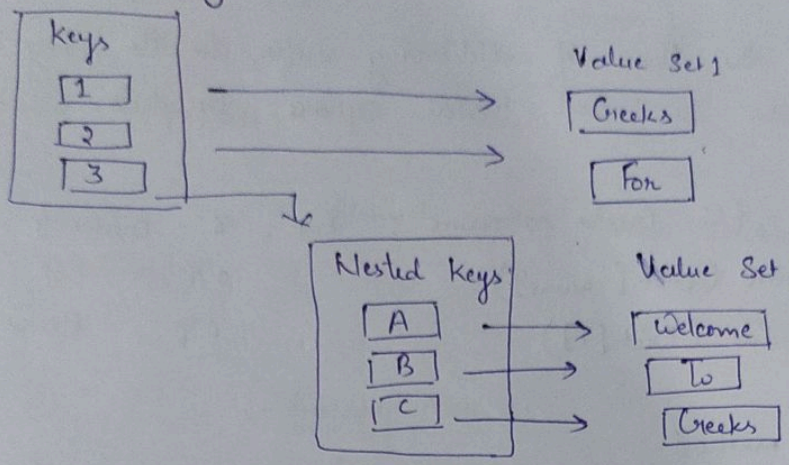
Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}
print(Dict)

Create using built-in fun" dict()

Dict = dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})
print(Dict)

with each item as a pair
Dict = dict([(1, 'Geeks'), (2, 'For')])
print(Dict)

Nested Dictionary :




```
Dict = { 1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': To,
                                   'c': 'Geeks'} }
Print (Dict)
```

Adding elements to a Dictionary

```
Dict = { }
Print (Dict)

# Adding elements one at time
Dict [0] = 'Geeks'
Dict [2] = 'For'
Dict [3] = 1
Print (Dict)

# adding set of values to a single key
Dict ['Value_set'] = 2, 3, 4
Print (Dict)
```

Accessing Elements from a Dictionary:

To access the items of dictionary refer to its key name. Key can be used inside square brackets

```
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}
print (Dict['name'])        O/P: For
print (Dict[1])             O/P: Geeks
```

get() method

```
print (Dict.get(3))         O/P: Geeks
```

Accessing element of a Nested Dictionary

```
Dict = {'Dict1': {1: 'Geeks'}, 'Dict2': {'Name': 'For'}}

print (Dict['Dict1'])               O/P: {1: 'Geeks'}
print (Dict['Dict1'][1])                 Geeks
print (Dict['Dict2']['Name'])            For
```

Removing Elements from Dictionary

```
del Dict[6]        # deleting a key Value
del Dict['A'][2]   # Nested
```

Dict.pop(1)

popitem() method

```
Dict = {1: 'Geeks', 'name': 'for', 3: 'Geeks'}
pop_ele = Dict.popitem()
print("The " pop_ele)
```

clear() : all items deleted from Dictionary

```
Dict.clear()
```

Dictionary Methods

copy() not take Parameter

```
dict1 = {1, 2, 3}
dict2 = dict1.copy
print(dict2)
```

File operations

↓

Read / Write to Files

1) JSON 2) TSV 3) CSV 4) Parquet 5) Txt

Read CSV file:
- CSV Module (Import csv)
- Pandas library (Import Pandas as Pd)

Method 1: csv.reader

(with: handles exception hand & no need to write close file)

```
import csv
with open ('ceo.csv', mode = 'r') as file :
    csvfile = csv.reader(file)
    for lines in csvfile:
        print (lines)
```

Method: to get data in Key - Pair

csv.DictReader()

Same as above csv.reader → csv.DictReader()

Method: Import Pandas → Method ↓

```
csvFile = pandas.read_csv('ceo.csv')
print (csvFile)
```

Read JSON file :

json support data types array, bool, int, float, string

Syntax: json.load (file object)

```
import json
f = open ('data.json',)
data = json.load (f)
# iterating through the JSON list
for i in data ['emp_details']:
f.close () # to close the file
```

Json.loads() : Reading from both string & JSON file.
→ to convert json data into Dictionary

```
import json
# json string
a = '{"name": "Bob", languages": "English"}'
```

→ it is json format

```
y = json.loads() # deserializes into dict & return dict.
print ("JSON string = ", y)
print()
f = open ('data.json', "r")
data = json.loads (f.read())
for i in data ['emp_details']:
    print (i)
```

f.close()

Reading & Writing to text files :-

Two types of files

1) Text files

2) Boolean (for ing) type data

```
import json
a = {"name": "Vivek", "age": 21}
print (type(a))
print (a)
a = json.dumps(a)
print (type(a))
```

→ returns string
Convert dict to string

O/P! Error.

~~a = json.~~

```
print (json.loads(a))
```

O/P!

→ opposite of dumps()

loads() and dumps() (Imp)

to convert json data into dictionary.

convert dictionary data into json

File Handling :

to handle file, we need to store somewhere it is imaginary, not a object

Pycharm
Create txt file
Create Py
↓
Very Simple

```
f = open('MyData', 'r')
print(f)
print(f.read())
print(f.readline())
print(f.readline())
```

→ to read line, default print only first line

It print second line

O/P: It makes space blw two line because in text another one is

```
print(f.readline(), end = "#")
```

due to print fun" (2 line gap)

→ to remove gap in line only one line gap as in text file

```
print(f.readline(4), end "#")
```

O/P: print only 4 chars of first line

To work

Write file :

```
f1 = open('abc', 'w')
f1.write("Something")
f1.write("People")
```

'w' : fist check then is a file 'abc' if not then it create

O/P: No output on consol but in file 'abc' that has been created

if (del that line) it automatically del from file also.

O/P: SomethingPeople

to add data in file, use append 'a'

~~f1.write('Mobile', 'a')~~ → to append

```
f1.open('abc', 'a')
f1.write('Mobile')
```

to add abc's file data to 'MyData' file

~~if open('abc'~~

```
f = open('MyData', 'r')
f1 = open('abc', 'w')
for data in f:
    f1.write(data)
```

To Read Img use boolean (binary) [Jpg]

```
f = open('Img-6309.jpg', 'r')
for in in f:
    print(i)
```

O/P: Error charmap as img is in binary format

In file there is two Mode

- Char
- Binary

to read file in binary format use 'rb'

```
f = open('Img-6309.jpg', 'rb')
for i in f:
    print(i)
```

O/P: we get hexavalue of img

to get img

```
f = open('Image-6309.jpg', 'rb')
f = open('MyPic.jpg', 'wb')
for i in f
    print(i)
f1.write(i)
```

O/P: another file Created MyPic.jpg and img see

a = 5

b = 0

try:

print (a/b) → only Run when error

except Exception:

print ("Hey, You can not divide a Number by zero")

print ("Bye")

If you want to print message Errors

a = 5

b = 0

try:

Print (a/b) is just object of representation

except Exception as e:

print ("Hey, You can not divide a Number by zero", e)

whenever we open file or resource, we need to close

a = 5

b = 0

try:

print ("resource open")

print (a/b)

print ("resource closed") #

except Exception as e:

print ("Hey, You cannot divide a Number by Zero", e)

b = 2 (also)

Finally block will be

if we

as

finally:

print ("resource closed") # remove closed"

Exception Handling: (Telusko video on youtube)

Syntactical Mistake shows compile time Error
Logical Error create Wrong Output
Run time Error:
e.g. divide by zero

```
a = 5
b = 2
print (a/b)
print ("Bye")
```

Statement
- Normal - will not give any error
- Critical

I/P : 2.5
Bye

```
a = 5
b = 0
print (a/b)
print ("Bye")
```

Error : as 0 not divided, and print ("Bye") not run

To avoid this we use exception handling: to run this statement

```
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 can not 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")
```

Study from geek for geeks also

Functions (Imp)

Why do we need fun" ?

We can create Multiple fun"s and Call when we need

fun" name

```
def greet():
    Print ("Hello")
    print ("Good Morning")
greet() ← Call fun"
```

fun" can be of two types

1) Return Something
ex: add

```
def add (x, y):
    c = x+y
    Print (c)
add (5,4)
```

```
def add (x,y):
    c = x+y
    return c
result = add(5,4)
Print (result)
```

```
def add_sub (x,y):
    c = x+y
    d = x-y
    Return c,d
result = add_sub(5,4)
Print (result)
result1, result2 =
        add_sub (5,4)
Print (result1, result2)
```

Functions Arguments in Python :-

```
def update(x):
    x = 8
    Print(x)
update(10)
```

O/P: 8

```
def update(x):
    x=8
    Print("x", x)
a=10
update(a)
print("a", a)
```

O/P:

x 8
a 10

passing a value a=10 not passing variable

we update value of x its not effect on value of a

Whenever you are calling a function by passing a Variable Value it will be pass as a Value not variable [ex:

- Pass By Value : means simply pass Value not address
- Pass By ~~Val~~ Reference: passes address itself, means you are not Passing 10, you are Passing address of a which means when you chage the value of x will affect the value of a this is Called Pass by Reference or Call by Reference.

not for Python

Everything in Python is an Object