

## **python - vijay sir notes**

### **Chapter 01: PYTHON BASICS =====**

#### **===== 01 Introduction =====**

Python:

- 
- 1)why Python?
  - 2)who can learn Python?
  - 3)How Python was Built?
  - 4)Python Real Time Applications
  - 5)Python Usage
  - 6)Python Features
  - 7)Python Comparision with other programming
  - 8)Python Scope and Jobmarket.
  - 9)Python Software Foundation(PSF)

---

why Python?

Python is a

- General Purpose
- Multi Programming paradigm
- High-level
- interpreted
- user-friendly programming language

Python is a user-friendly programming language which provides simple syntaxes which looks like normal english stmts.

Python was developed by Guido Van Rossum in the year 1989.

---

Python was built or derived by taking the features or advantages from various other programming languages

such as

- 1)Procedural Oriented PRogramming: ex :C-Language----->Functions
- 2)Object Oriented Programming           ex: C++,Java       ----->Security,Re-usability,flexibility
- 3)Scripting Languages                   ex: shell script----->dynamic Datatypes
- 4)Modular PRogramming                ex: modula-3       ----->Modules----->89300

---

Dynamic Datatypes:

| C/C++/Java              | Python   |
|-------------------------|--|
| int x=10<br>dynamically | x=10 based on the value assigned the variables are created |
| float y=4.5             | y=4.5 type(y)---->float                                    |
| string z="hello"        | z="hello"  |

---

### Modules :

python provides modules for each and every domain such as

DataScience ----->seperate modules

```
MACHINE Learning----->      "
BigData----->      "
Testing ----->      "
Networking ----->      "
Animations----->      "
OS ----->      "
Oracle----->      "
mysql----->      "
excel ----->      "
xml ----->      "
CSV----->      "
pdf ----->      "
math----->      "
statistics----->      "
Administrative Activities>      "
Graphs----->      "
```

---

### Python Real Time Applications:

Today python is mostly used in developing the following Applications:

Python Real Time Applications:

1. Web Applications:

2. Automation Applications

- DataScience----->MACHINE Learning--->
- IOT

3. Bigdata: KB, MB, GB, TB, PB, EB, ZB, YB, SB

4. Scientific Applications:

5 .GUI/Animations :

6. Game Development: PyGame,Pykara ,PySoy

7. Software Development:

8. Business : Tryton

9. Database

10. N/W Programming :Twisted Python

11. Audio and video Applications

12. CAD Applications

13. Embeded Applications

14. Desktop Applications:

15. WebScraping:

16. Computer vision

17. Robotics

18. Data Analysis

---

who are using Python Today:

1) Google : USes Python Script in web Search systems.

2) YouTube : Video sharing service is largely written in python.

3) MMOG : Massively Multi-player Online Games uses python

4) Maya : A powerful 3D animation system provides python scripting API

5) iRobot: uses python to develop commercial and military robotic devices.

6) BitTorrent: File sharing system, began its life as a python program.

7) Google's App Engine: webdevelopment framework uses python as an application language.

8) Ironport Email server : USes more than 1 million lines of python code to do its job.

9) HArdware Testing: Intel,Cisco,HP,Seagate,Qualcomm ,IBM etc uses python for H/W Testing.

10) Financial MArketing.....

11)Yahoo

12)Instagram

13)Quora

14)Reddit

15)Pinterest

16)Dropbox

---

#### Python Features:

1)Powerfull standard Library: provides many Builtin Functionalities for performing various operations.

2)Wide varieties of modules :89300 modules

3)Supports interactive mode: USer can easily interact with Python interpreter line by line.

4)Supports dynamic datatypes:

5)supports Object oriented PRogramming:

6)supports all major databases.

7)Extendable

8)Can be easily integrated with other languages

9)simple and easy and looks like normal english stmnts.

---

#### Python Comparision with other PRogramming:

ex: accepting 2 no's from the user and performing addition.

br.

1)Using Java

```
import java.io.*;
class sample
{
    public static void main(String args[]) throws IOException
    {
        InputStreamReader isr=new InputStreamReader(System.in);
```

```
BufferedReader br=new BufferedReader(isr);
System.out.println("Enter 1st No:");
String s1=br.readLine();
int i=Integer.parseInt(s1);
System.out.println("Enter 2nd No:");
String s2=br.readLine();
int j=Integer.parseInt(s2);
System.out.println(i+j);
}
}
```

---

## 2)Using Python:

```
x=int(input("Enter value of x:"))
y=int(input("Enter value of y:"))
print(x+y)
```

---

## Python Software Foundation(PSF):

PSF is a non-profit organization devoted to python programming language.

-It was launched on March 6th,2001.

-Mission : to faster the development of python community.

-in 2005, PSF received "COMputerworld Horizon Award" for cutting-edge Technology

President and Founder : Guido Van Rossum

Headquarters: Delaware,Unitedstates

Official website: [www.python.org/psf](http://www.python.org/psf)

---

## Python versions:

python 1.xxx ----->1989  
python 2.xxx ----->2000  
python 3.xxx ----->2008

Latest version of python----->python 3.13  
Stable version of python----->python 3.12  
(mostly used version)

---

## Python Installation:

Google----->python download----->download---->all releases---->select python 3.12

click 64 bit executable installer--->download starts

dblclick the downloaded file----->select add python to path

go for customize installation----->next---->select location to install as  
C:\python313 --- install

check whether python is installed correctly or not:

goto cmdprompt----->type python----->  
it should display python version and interpreter.

---

Python IDE's

- 1)Pycharm----->mostly used
  - 2)Pydev
  - 3)Komodo
  - 4)Spyder
  - 5)Eclipse
  - 6)NEtbeans
  - 7)Jupyter Notebook
  - 8)VSCode
  - 9)Atom
- 

Python Execution Modes: 2 modes of Execution

- 1)Interactive mode
- 2)Batch mode

Interactive mode: Here we directly type our python stmts on interpreter and can easily communicate line by line.

Advantage: For learning Python and to test the functionalities of python

Batch mode: Here we write set or batch of stmts within Editors/IDE's and save it by using .py extension and later submit to python interpreter.

various Editors

- 1)Notepad
- 2>Editplus
- 3)vi
- 4)nano
- 5)gedit

Various IDE's (Integrated Development Environment)

- 1)Pycharm----->mostly used
  - 2)Pydev
  - 3)Komodo
  - 4)Spyder
  - 5)Eclipse
  - 6)Netbeans
  - 7)Jupyter Notebook
  - 8)VSCode
  - 9)Atom
- 

Developing Python Applications using Editors

ex:Notepad

open notepad and type the following python stmts.

```
x=10  
y=20  
print(x+y)  
print(x-y)  
print(x*y)  
print(x/y)
```

save it using sample1.py extension and later submit to python interpreter.

submitting the file to interpreter----->

goto cmdprompt--->python C:\python8ambatch\sample1.py

It gives the output

here we are writing the code at one place and executing it at other place instead we can write and execute at a single place, for that whenever we install python, along with python we can IDLE(Integrated Development Environment)---->where we can work with interactive mode as well as batch mode.

---

Datatypes: 2 Types

1.Fundamental Datatypes----->Object holding a single value

- 5Types
  - 1.int ---->age=25
  - 2.float --->sal=90870.40
  - 3.boolean-->True/False
  - 4.string -->city="Hyderabad"
  - 5.complex-->x+jy
-

2. Collection Datatypes -----> Object holding group of values

4Types

## 02 Datatypes

## '''Fundamental Datatypes:

- i)int
  - ii)float
  - iii)boolean
  - iv)complex
  - v)string

## #working with integers

x=10  
y=20

```
print(x)  
print(type(x))  
print(id(x))
```

```
print(y)  
print(type(y))  
print(id(y))
```

```
z=x+y  
print(z)  
print(type(z))  
print(id(z))
```

## # fundamental Datatypes:

#1.int

#case 1:

```
x=10
```

```
print(x)
print(id(x))
print(type(x))
```

---

```
#case 2: Defining 2 variables with 2 different values--->2 objects are created with 2 different addresses
```

```
x=10
```

```
y=20
```

```
print(x)
print(id(x))
print(type(x))
```

```
print(y)
print(id(y))
print(type(y))
```

```
z=x+y
print(z)
print(id(z))
print(type(z))
```

```
w=y-x
print(w)
print(id(w))
print(type(w))
```

---

```
# case 3: defining 2 variables with same values---->only one object created and its address is returned to both variables
```

```
x=10
```

```
y=10
```

```
print(x)
print(id(x))
print(type(x))
```

```
print(y)
print(id(y))
```

```
print(type(y))
```

```
z=x+y  
print(z)  
print(id(z))  
print(type(z))
```

---

```
#case 4:initializing multiple values to a variable---> multiple objects created--->always latest value will be returned.
```

```
x=10  
print(id(x))
```

```
x=20  
print(x)  
print(id(x))  
print(type(x))
```

```
x=10  
print(x)  
print(id(x))  
print(type(x))
```

---

```
# float datatype  
x=4.5
```

```
print(x)  
print(type(x))  
print(id(x))
```

```
y=20  
print(y)  
print(id(y))  
print(type(y))
```

```
z=x+y  #int+float----->float  
print(z)  
print(type(z))
```

```
print(id(z))
```

## ===== 03 Strings =====

#String :A String is a sequence collection of characters

#In Python, a string is represented using'

- 1.Single quote(' )
- 2.Double quote(" ")
- 3.Triple quote(""" """)

```
city="hyderabad"
```

Each character of a string is represented or accessed by a unique index

String supports 2 types of indexes:

1.Forward index or Positive index-->index starts from left to right  
->index starts with 0

2.Backward and Negative index----->index starts from right to left  
-->index starts with -1

.....

```
x="python"  
print(x)  
print(type(x))  
print(len(x))
```

```
y="programming"  
print(x+" "+y)#Concatenation-->adding 2 strings-->1st string followed by 2nd string
```

---

#String methods:

#1.upper():To print in upper case  
x="hyderabad"  
print(x.upper())

#2.lower() :To print in lowercase  
x="MUMBAI"  
print(x.lower())

```

#3.capitalize() :For capitalizing the 1st character of a string
x="india"
print(x.capitalize())

#4.title(): For capitalizing the 1st character of each word in a string
x="good evening hyderabad"
print(x.title())

#ex:
emps="rohit,ajay,miller,james,amar,blake,john,ankith"
print(emps.title())

#5.swapcase :For converting one case to another
x="good evening hyderabad"
print(x.swapcase())

#6.replace():For replacing a portion of a string
x="Java is Easy and Java is simple"
print(x.replace("Java","Python"))

#7.count():To count the no of occurrences of a sub-string or a character
x="hello hello hello...how are you??"
print(x.count("hello"))
print(x.count("h"))

data="""
1. Python is Simple
2. Python is user-friendly
3. Python supports interactive mode
4. Python supports 89300 modules
5. Python has many built-in libraries
6. Python supports object-oriented programming
7. Python supports all major databases
8. Python is dynamic typed
9. Python provides simple syntaxes
10. Python is extensible.
"""

print(data.count("Python"))
print(data.count("python"))
print(data.lower().count("python"))

#8.format() :for substitutions
x("{} is the captain of Indian Team"
print(x.format("Rohith"))

#ex:2
y("{} is the {} of {}"
print(y.format("Harish","son","Lakshmi"))
print(y.format("Modi","PM","India"))

```

```
print(y.format("Python","most popular","all languages"))

#9.strip():for removing the blank spaces before and after the string
x="           Good Evening Hyderabad           "
print(x,len(x))

y=x.strip()
print(y,len(y))

#10.lstrip():leftmost strip
y=x.lstrip()
print(y,len(y))

#11.rstrip():rightmost strip
y=x.rstrip()
print(y,len(y))

#12.find():To check whether a sub-string is available or not
#If found, it returns the 1st character index position of the sub-string
#else it returns -1
x="Python is easier than Java "
print(x.find("Java"))
print(x.find("Python"))
print(x.find("hadoop"))

#13.rfind(): right most find
x="Python is easier than Java and Python is simpler than Java"
print(x.rfind("Java"))
print(x.rfind("Python"))
print(x.rfind("hadoop"))

#14.split() :If we split a string,we get a list
x="Good Evening Hyderabad"
#Task: print---->"hyderabad"
y=x.split(" ")
print(y,type(y))
print(y[2])

#ex:2
x="101,Ajay,50000,Manager"
#Task--->print--->name and designation
y=x.split(",")
print(y,type(y))
print(y[1],y[3])
```

---

```
#String slicing: Extracting particular elements from a list

x="python program"
#Extract--->"python"
print(x[0:6])#Always upperbound is excluded

#Extract-->"prog"
print(x[7:11])
print(x[11:7])#Empty string-->bcoz always slicing in the forward direction
#Extract-->"prog" using -ve index
print(x[-7:-3])
#Extract--->"program"
print(x[7:14])
print(x[7: ])
#Extract--->"program" using -ve index
print(x[-7: ])
print(x[ :6])
print(x[ : ])
print(x[ :])
print(x)
print(x[0: ])
```

===== 04 output stmt =====

#print():output function or output stmt

```
x=10
y=20
print(x,y)
print(x,",",y)
print()
print(x,y,sep=",")
print(x,y,sep=" ") ; print(x,y,sep="\t")
print(x,y,sep=":")
print("\n")
print(x,y,sep="-->")
print(x,y,sep="\n")

a=10;b=20;c=30
print(a);print(b);print(c)
```

```
print(a,b,c,sep=" ")
print("Good",end=" ")
print("Morning")

#ex:
print("Good",end="\t")
print("Morning")

#ex:
print("Good",end="")
print("Morning")

#ex:
print("Good",end=" ")
print("Morning",end=" ")
print("Hyderabad")
```

=====

#print():output function or output stmt

```
x=10
y=20
print(x,y)
print(x,",",y)
print()
print(x,y,sep=",")
print(x,y,sep=" ") ; print(x,y,sep="\t")
print(x,y,sep=":")
print("\n")
print(x,y,sep="-->")
print(x,y,sep="\n")
```

```
a=10;b=20;c=30
print(a);print(b);print(c)
print(a,b,c,sep=" ")
```

```
print("Good",end=" ")
print("Morning")
```

```
#ex:
print("Good",end="\t")
print("Morning")
```

```
#ex:  
print("Good",end="")  
print("Morning")
```

```
#ex:  
print("Good",end=" ")  
print("Morning",end=" ")  
print("Hyderabad")
```

```
=====
```

```
#print():output function or output stmt
```

```
x=10  
y=20  
print(x,y)  
print(x,",",y)  
print()  
print(x,y,sep=",")  
print(x,y,sep=" ") ; print(x,y,sep="\t")  
print(x,y,sep=":")  
print("\n")  
print(x,y,sep="-->")  
print(x,y,sep="\n")
```

```
a=10;b=20;c=30  
print(a);print(b);print(c)  
print(a,b,c,sep=" ")
```

```
print("Good",end=" ")  
print("Morning")
```

```
#ex:  
print("Good",end="\t")  
print("Morning")
```

```
#ex:  
print("Good",end="")  
print("Morning")
```

```
#ex:  
print("Good",end=" ")  
print("Morning",end=" ")  
print("Hyderabad")
```

===== 05 input stmt =====

```
#input(): for accepting input from the user keyboard
```

```
fname=input("Enter the first Name:")
lname=input("Enter the last Name:")
print(fname+" "+lname)
print(fname,lname,sep=" ")
print(fname,lname)
```

---

```
#accepting 2 values from the user keyboard and performing addition
#Note:whatever we are accepting through input function by default
#      taken as string and later we need to convert(typecast) to required type
#Typecasting functions--->int(),float(),str(),bool()
```

```
x=int(input("Enter value of x:"))
y=int(input("Enter value of y:"))
print(x,type(x))
print(y,type(y))
print(x+y)
```

---

```
#Accepting customer details
```

```
cname=input("Enter the Customer Name:")
accno=int(input("Enter the account No:"))
bal=float(input("Enter the balance:"))

print("CUSTOMER NAME=",cname)
```

```
print("ACCOUNT NO=",accno)
print("BALANCE=",bal)

print("\n")
print("CUSTOMER NAME=",cname,"\\nACCOUNT NO=",accno,"\\nBALANCE=",bal)
```

===== 06 ifelse =====

#Flow-control stmts

-Every Python stmt executes only once from top to bottom

#or  
-In-order to execute or not to execute set of stmts,we go for  
flow-control stmts  
or  
-To execute set of stmts for repeated no of times or for multiple times  
we go for loops or control structures

#Flow-control stmts are divided into 2 parts

1.condition:

-an expression which returns a boolean value

2.Block/clause: set of stmts following the same space indentation

#Various flow-control stmts/loops are:  
1.if  
2.if-else  
3.elif  
4.for  
5.while  
6.while-else  
7.break  
8.continue  
9.pass

#Syntax for defining if-else

```
if(condition):
    stmt1
    stmt2---->Block1
    stmt3
    stmt4
else:
    stmt5
```

```
stmt6---->Block2  
stmt7  
stmt8
```

If the condition is True, then the stmts within the if-block(block1) executes  
If the condition is False then the stmts within the else-block executes

#Program illustrating if-else

```
x=int(input("Enter value of x:"))
```

```
if(x<=4):  
    print("ONE")  
    print("TWO")  
    print("THREE")  
    print("FOUR")  
else:  
    print("FIVE")  
    print("SIX")  
    print("SEVEN")  
    print("EIGHT")
```

#To check whether 2No's are equal or not

```
x=int(input("Enter value of x:"))
```

```
y=int(input("Enter value of y:"))
```

```
if(x==y):  
    print("No's are equal!!!")  
else:  
    print("No's are not equal!!!")
```

#To check whether a given number is a even number or not

```
x=int(input("Enter value of x:"))
```

```
if(x%2==0):  
    print(x,"is a Even number")  
else:  
    print(x,"is a odd number")
```

```
#elif:For providing multiple conditions

#Program illustrating the change in time

time=float(input("Enter the current time:"))

if(time<=12.00):
    print("Good Morning!!!")
elif(time<=16.00):
    print("Good Afternoon!!!")
elif(time<=20.00):
    print("Good Evening")
else:
    print("Good Night!!!")
```

===== 07 for loop =====

```
#For loop:
#For loop executes set of stmts for every element of collection object
```

#All the stmts within the for loop should follow the indentation

```
x=[10,20,30,40,50]
```

```
for p in x:
    print(p)
```

#printing all the values in a single line

```
for p in x:
    print(p,end=" ")
```

#printing a msg for multiple times

```
x=[10,20,30,40,50]

for p in x:
    print("Good Morning!!!")

#Squaring each element of a list

x=[10,20,30,40,50]

for p in x:
    print(p*p) #or print(p**2)
print("end")

#for loop on a string
x="python"

for p in x:
    print(p)

#printing each character for 3 times

for p in x:
    print(p*3)

#printing a msg for multiple times
for p in x:
    print("Good Evening")

#range(n):Generates range of values from 0 to n-1
#range(10): generates values from 0 to 9

x=range(10)
print(x)
print(type(x))

#applying for loop and   getting one by one   value from the range object
for p in x:
    print(p)

#ex:2
y=range(10,20)
```

```
for q in y:  
    print(q)  
  
#ex:3  
z=range(20,30,2)  #(Startvalue,stopvalue,stepvalue)  
for r in z:  
    print(r)  
  
#ex:4  
w=range(40,30,-1)
```

```
for s in w:  
    print(s)
```

```
#range(n):Generates range of values from 0 to n-1  
#range(10): generates values from 0 to 9
```

```
for p in range(10):  
    print(p)
```

```
for q in range(10,20):  
    print(q)
```

```
for r in range(20,30,2):  
    print(r)
```

```
for s in range(40,30,-1):  
    print(s)
```

```
#printing a msg for 100 times
```

```
for p in range(1,101):  
    print(p,"Good Evening!!!")
```

```
#printing numbers from 1 to 300
```

```
for p in range(1,301):  
    print(p,end=" ")
```

```
#printing only even numbers upto 200
```

```
for p in range(2,201,2):  
    print(p,end=" ")
```

```
#printing only odd numbers upto 200
```

```
for p in range(1,201,2):  
    print(p,end=" ")
```

```
===== 08 while loop =====
```

```
#while loop :
```

```
#while loop executes set of stmts for repeated number of times or for  
#multiple times
```

```
#Syntax for defining a while loop
```

```
:::::
```

```
initialization
```

```
while(condition):
```

```
    stmt1
```

```
    stmt2
```

```
    stmt3
```

```
    stmt4
```

```
    increment/decrement
```

The stmts within the while loop will keep on executing until the condition becomes false,

once the condition becomes false then the ctrl comes out of while loop and executes the other stmts

```
:::::
```

```
#printing nos from 1 to 10
```

```
x=1
```

```
while(x<=10):
```

```
    print(x)
```

```
    x=x+1
```

```
print("end")
```

```
#printing a msg for multiple times using while loop
x=1
while(x<=10):
    print("Good Evening!!!")
    x=x+1
print("End")
```

```
#Infinite loop
```

```
x=1
while(True):
    print(x,"hello")
    x=x+1
```

```
#break stmt: If break stmt is used in a loop then ctrl comes out of that loop
#           and stmts after break are not executed
```

```
x=1
while(True):
    print("hello")
    if(x==5):
        break
        print("hi")
    x=x+1
print("end")
```

```
#while-else:
#If while condition becomes false then control goes to else block
```

```
x=1
while(x<=10):
    print("hello")
    x=x+1
else:
    print("Good Morning!!!")
print("End")
```

```
#continue :  
#It skips the current iteration and continues with the next iterations  
  
x=0  
while(x<10):  
    x=x+1  
    if(x==5):  
        continue  
        print("hi")  
    print(x,"hello")  
print("end")
```

```
#break and continue  
  
while(True):  
    username=input("Enter the username:")  
    if(username!="vijay"):  
        continue  
    password=input("Hello....Vijay!!!...please enter ur password:")  
    if(password=="python"):  
        break  
print("Login successfull!!!!")
```

```
#pass :represents an empty block  
  
x=10  
y=5  
if(x>y):  
    pass  
else:  
    pass
```

## ===== 1 files intro =====

#File Handling:

All programming languages process data in the Ram, but RAM is a volatile, once execution is finished, the data is removed.

Programming Languages are good for processing  
Programming Languages are bad for storage

python programs are processed within ram and after processing they can be stored in any of the following.

- 1) Files--->.txt,.csv,.xls,.pdf,.xml,.json
- 2) Databases-->oracle,mysql,DB2
- 3) NoSQL--->cassandra,mongoDB,HBase,

python can easily connect with the files and databases to store and retrieve the data.

The various operations we can perform on a file:

1. opening a file
2. closing a file
3. Reading data from a file
4. writing data to a file, etc...

for performing above operations, we have built-in functions.  
such as open(), close(), read(), write() etc...

## ===== 2 modes =====

open(): open function is a pre-defined function used to open a file

At the time of opening a file, we need to specify the mode of the file.  
mode indicates for what purpose you are opening the file.

The various modes are

| Mode         | Description  |
|--------------|--|
| 1. 'r'-----> | opens a file for reading data(default mode)  |
| 2. 'w'-----> | opens a file to write data into a file.<br>if file already exists then it truncates(removes) and writes.<br>if file doesn't exist, it automatically creates a new file and writes into it. |
| 3. 'a'-----> | opens a file to append data at the end of the file.<br>if file doesn't exist it creates a new file and appends<br>if file already exists, it appends the data.                             |

4. 't'-----> opens a file in text mode.
  5. 'b'----->opens a file in binary mode.
  6. 'w+'----->opens a file to write and read data of a file.  
The previous data in the file will be deleted.
  7. 'r+'----->opens a file to read and write data into a file.  
The previous data in the file will not be deleted.  
The File pointer will be at the beginning of a file.
  8. 'a+'----->opens a file to append and read data of a file.  
The file pointer will be at the end of the file.
- 

read mode: for reading data from a file, we have 3 methods

- 1.read() :It reads the entire file and returns ---->multi-line string
- 2.readline() :It reads a particular line and returns --->single-line string
- 3.readlines() :It reads all the lines of a file and creates a list  
Each line will be treated as a element of a list.

1Q. Python code to read a text file

2Q. Python code to read a CSV file

3Q. Python code to read a java file

4Q. Python code to read a python file

5Q. Python code to read a Excel file

6Q. Python code to read a PDF file

7Q. Python code to read a particular line in a file

8Q. Python code to read 2nd half of the line

9Q. Python code to read last 5 lines from a file

10Q. Python code to read first 5 lines from a file

11Q. Python code to specific lines of a file

12Q. Python code to create a text file and write data into it

13Q. Python code to create a CSV file in particular location

14Q.Python code to create a Java file

15Q.Python code to create a python file and write data into it

16Q.Python code to create a Excel workbook

17Q.Python code to create headers within the Excel file

18Q.Python code to create multiple sheets within a Excel

19Q.Python code to count the number of rows and columns in a Excel file

20Q.Python code to read a particular column within a Excel

21Q.Python code to read a particular row within a Excel

22Q.Python code to read all the rows within a Excel

23Q.Python code to append data to a existing file

24Q.Python code to read a file and write that data into another file

25Q.Python code to read multiple files and write to a single file

===== 3 open read close =====

```
f=open("demo1.txt",mode="r") #whenever we call open() function then  
#a file object is created  
#here file object 'f' is created for reading purpose
```

```
print(f.read())
```

```
f.close()
```

```
f=open("demo1.txt",mode="r")  
  
#without using read() method, displaying the content of the file  
  
for p in f:  
    print(p)
```

```
f1=open("demo1.txt",mode="r") #whenever we call open() function then
```

```

#here file object 'f' is created for reading purpose

f2=open("C:/python/sample.java","r") #Reading a java file

f3=open("C:/python/sample2.py")      #Reading a .py file

f4=open("C:/data1/emp.csv",mode="r") #Reading a .csv file
print(f1.read())
print(f2.read())
print(f3.read())
print(f4.read())
f1.close()
f2.close()
f3.close()
f4.close()

f1=open("demo1.txt",mode="r")
print(f1.read())
f1.close()

f2=open("C:/python/sample.java","r") #Reading a java file
print(f2.read())
f2.close()

f3=open("C:/python/sample2.py")      #Reading a .py file
print(f3.read())
f3.close()

f4=open("C:/data1/emp.csv",mode="r") #Reading a .csv file
print(f4.read())
f4.close()

```

#### Python features:

1. powerfull Standard Library: Introduced many builtin functionalities.
2. Wide variety of modules : python supports nearly 89,300 modules
  - Seperate module for Networking
  - Seperate module for Testing
  - Seperate module for Data Analytics
  - Seperate module for GUI/Animations
  - Seperate module for OS etc....so on...
3. portable: can run on any h/w platforms.
4. Interpreted : Python is not compiled,it is interpretation language like PERL and PHP

5. Object-Oriented :Supports OOPs features
6. Extendable : Adding various types of modules.
7. Database Support: Provides interface to all major databases.
8. simple : Easy to develop applications in Python
- 9.GUI Programming: supports GUI applications
10. Interactive mode : supports interactive mode for testing and debugging
11. Supports dynamic datatypes.
- 12.can be easily integrated with other languages like C,C++,java,CORBA etc..

===== 5 tell seek =====

file name: sample.txt:  
welcome to hyd.....  
welcome to pune.....  
welcome to chennai.....

#tell() : tells where currently the cursor is  
#seek() : moving the cursor to the desired position.

```
f=open("sample.txt")
print(f.tell()) #tells the current cursor position
print(f.read())
print(f.tell())
f.seek(8) #moving the cursor to desired position
print(f.tell())
print(f.read())
print(f.tell())
f.seek(11)
print(f.read(4)) #reads 4 characters from current position
print(f.tell())
f.close()
```

===== 6 tell read seek=====

file name: hello.txt:  
hello hyderabad

```
hello mumbai  
hello chennai
```

file name: hello1.txt:  
hello world  
hai pune

```
#tell(),read(),seek()
```

```
f=open("hello.txt")  
print(f.tell())  
print(f.read())  
print(f.tell())  
f.seek(f.tell()-7)  
print(f.tell())  
print(f.read().upper())  
print(f.tell())  
f.close()
```

```
f=open("hello1.txt")
```

```
s1=f.read()  
print(s1,type(s1),len(s1))  
print(s1.find("pune"))  
print(f.tell())  
f.seek(f.tell()-5)  
print(f.tell())  
print(f.read())
```

===== 7 readline =====

file name: sample1.txt:  
hello hyderabad  
hello mumbai  
hello chennai

```
#readline():reads a line and ctrl goes to next line
```

```
f=open("sample1.txt",mode="r")  
print(f.tell())  
print(f.readline())  
print(f.tell())  
print(f.readline())  
print(f.tell())
```

```

print(f.readline())
f.seek(5)
print(f.readline())
print(f.tell())
print(f.read(5))
print(f.readline())
print(f.tell())
print(f.read())

f.close()

#read/print-->only the 3rd line of a file using readline
f=open("sample1.txt",mode="r")

f.readline()
f.readline()
print(f.readline())

f.seek(0)
for i in range(1,3):
    f.readline()
print(f.readline())


#sample1.txt
#####
hello hyderabad
hello mumbai
hello chennai
#####

#print-->"mumbai" from the file
f=open("sample1.txt",mode="r")
x=f.read()
print(type(x))

print(x.find("mumbai"))
f.seek(x.find("mumbai"))
print(f.readline())

#in the above file-->modify the city-->"chennai" to "pune"

y=x.replace("chennai","pune")
print(y)

```

===== 8 readlines =====

file name: testdata.txt:

```
101,Miller,10000,11,Hyd  
102,Blake,20000,12,Pune  
103,Sony,30000,11,hyd  
104,Sita,40000,12,pune  
105,John,50000,13,hyd
```

file name: sample1.txt:

```
hello hyderabad  
hello mumbai  
hello chennai
```

#readlines: reads all the lines and creates a list

```
f=open("sample1.txt",mode="r")
```

```
x=f.readlines()  
print(x,type(x),len(x))
```

```
for p in x:  
    print(p)
```

```
#I want to print the 2nd line of a file using readlines()  
print(x[1])
```

```
#diff b/w read() and readlines()  
#both reads the entire file
```

```
#read()-->reads the entire file and returns a string(multi-line)  
#readlines()-->reads the entire file and returns a list as output  
#           no of elements in list= no of lines in file  
#           if 10 lines in a file--->then 10 elements in a list
```

"""\ testdata.txt--->present in C:/data1/testdat.txt  
101,Miller,10000,11,Hyd  
102,Blake,20000,12,Pune  
103,Sony,30000,11,hyd  
104,Sita,40000,12,pune  
105,John,50000,13,hyd  
"""

```
f=open("C:/data1/testdata.txt",mode="r")
#or f=open("testdata.txt",mode="r")
#Task :print only the empnames and salaries
x=f.readlines()
print(x)
print("\n")

for p in x:
    print(p.split(",")[1],p.split(",")[2])
    #print(p.split(",")[1:3])
```

----- 9 write mode -----

file name: sample1.txt:

1. Python is Simple
2. Python is user-friendly
3. Python supports interactive mode
4. Python supports 89300 modules
5. Python has many built-in libraries
6. Python supports object-oriented programming
7. Python supports all major databases
8. Python is dynamic typed
9. Python provides simple syntaxes
10. Python is extensible.

Good Evening....Hyderabad

Hello

file name: emp.CSV:

```
101,Ajay,50000,m,11
102,Sanjay,60000,m,12
103,Amar,70000,m,13
```

#Creating a text file and write data into it

```
f=open("C:\\python8pmbatch\\sample1.txt",mode="w")
#f=open("sample1.txt",mode="w")
#here file object f is created only for writing purpose
```

- ```
data=""1. Python is Simple
2. Python is user-friendly
3. Python supports interactive mode
4. Python supports 89300 modules
```

5. Python has many built-in libraries
6. Python supports object-oriented programming
7. Python supports all major databases
8. Python is dynamic typed
9. Python provides simple syntaxes
10. Python is extensible.

```
""  
f.write(data)  
f.write("\nGood Evening....Hyderabad")  
f.write("\nHello")  
f.close() #If we wont close a file,we cannot perform any operation  
#           either reading,writing etc
```

```
f1=open("C:\\\\python8pmbatch\\\\sample1.txt",mode="r")  
#f1=open("sample1.txt",mode="r")  
print(f1.read())  
f1.close()
```

#Creating a .csv file and writing data into it

```
f=open("emp.csv",mode="w") #File object f is created for writing  
""  
f.write("101,Ajay,50000,m,11")  
f.write("\n102,Sanjay,60000,m,12")  
f.write("\n103,Amar,70000,m,13")  
"  
#or  
emps=""101,Ajay,50000,m,11  
102,Sanjay,60000,m,12  
103,Amar,70000,m,13  
"  
f.write(emps)  
f.close()
```

```
f1=open("emp.csv",mode="r")  
print(f1.read())  
f1.close()
```

#creating a .py file and writing into a python file

```
f=open("test1.py",mode="w")  
""  
f.write("x=[10,20,30,40,50]\n")
```

```
f.write("print(x)\n")
f.write("print('sum of list elements=',sum(x))")
"""
data=""
x=[10,20,30,40,50]
print(x)
print("sum of list elements=",sum(x))
"""
f.write(data)
f.close()
```

```
f1=open("test1.py")
print(f1.read())
f1.close()
```

#writing into a python file to add 2 nos

```
f=open("test2.py",mode="w")

#f.write("x=20\n y=20\nprint(x+y)")
#or
data=""
x=int(input("Enter value of x:"))
y=int(input("Enter value of y:"))
print(x+y)
"""
f.write(data)
f.close()
```

```
x=[10,20,30,40,50]
print(x)
print("sum of list elements=",sum(x))
```

```
x=int(input("Enter value of x:"))
y=int(input("Enter value of y:"))
print(x+y)
```

===== 10 Accepting input from keyboard and writing into afile =====

file name: emps.txt:  
101 Ajay 70000 11  
102 Bhaskar 90000 12

```
#Accepting input from keyboard and writing into a file
```

```
f=open("emps.txt",mode="w+")

emp=input("Enter the emp details:")
f.write(emp)
f.seek(0)
print(f.read())
f.close()
```

```
#r+mode
f=open("emps.txt","r+")
print(f.read())
```

```
emp=input("Enter the emp details:")
f.write("\n")
f.write(emp)
f.seek(0)
print(f.read())
f.close()
```

===== 11 append mode =====

file name: emps.txt:

```
101,Ajay,50000,m,11
102,Rahul,60000,m,12
103,James,30000,m,11
104,Amar,40000,m,12
105,Rohith,50000,m,13
106,Ajith,60000,m,12
107,Ramesh,70000,m,11
108,Akhil,80000,m,12
109,Amar,90000,m,13
```

#append mode:

```
f=open("emps.txt",mode="a+")
f.write("\n103,James,30000,m,11\n104,Amar,40000,m,12")
f.seek(0)
print(f.read())
f.close()
```

#If the program executed for multile times then the same records will be appended

```
#multiple times
#Here each time we are appending same records,but i want to append different
#records in each execution

#difference B/W r+ and a+ --->both are appending data

#r+ --->reads and writes--->here without reading if we write
#                               it overwrites the data

#a+ ---->appends and reads
```

```
#append mode:
f=open("emps.txt",mode="a+")
x=input("Enter the new Emp details:")
f.write("\n")
f.write(x)
f.seek(0)
print(f.read())
f.close()
```

```
#appending multiple records
f=open("emps.txt",mode="a+")
n=int(input("Enter the nos of employees to be added:"))
for p in range(1,n+1):
    x=input("Enter the new Emp details:")
    f.write("\n")
    f.write(x)
f.seek(0)
print(f.read())
f.close()
```

===== 12 with stmt =====

file name: demo1.txt:

1. Python is Simple
2. Python is user-friendly
3. Python supports interactive mode
4. Python supports 89300 modules
5. Python has many built-in libraries
6. Python supports object-oriented programming
7. Python supports all major databases
8. Python is dynamic typed

9. Python provides simple syntaxes

10. Python is extensible.

```
#with stmt:  
#The with statement can be used while opening a file,  
#The advantage of with stmt is it takes care of closing a file which is opened by it  
#with stmt also follows space indentation  
  
data=""  
1. Python is Simple  
2. Python is user-friendly  
3. Python supports interactive mode  
4. Python supports 89300 modules  
5. Python has many built-in libraries  
6. Python supports object-oriented programming  
7. Python supports all major databases  
8. Python is dynamic typed  
9. Python provides simple syntaxes  
10. Python is extensible.  
"  
with open("demo1.txt", mode="w") as f:  
    f.write(data)  
  
with open("demo1.txt", mode="r") as f1:  
    print(f1.read())  
  
#or reading using for loop  
with open("demo1.txt", mode="r") as f2:  
    for line in f2:  
        print(line)
```

### ===== 13 Assignment tasks =====

1. Python code to read last 'n' lines from a file

2. Python code to read first 'n' lines from a file

3. Python code to read specific lines from a file i.e 3rd and 5th line

4. Python code to read data from one file and writing that data to other file

5.Python code to read multiple files and writing to a single file

6.Python code to read specific lines and write to another file

7.Python code for reading a file and adding some data and writing to another file

8.Python code to read data from a file and check the no of occurrences  
of a particular word

9.python code to read data from a particular file and convert the  
entire text to uppercase

10.python code to read data from a file and check whether a sub-string is  
available or not

11. python code to overwrite or modify the 3rd record/line in a file

===== 14 Reading n lines from a file =====

file name: demo1.txt:

1. Python is Simple
2. Python is user-friendly
3. Python supports interactive mode
4. Python supports 89300 modules
5. Python has many built-in libraries
6. Python supports object-oriented programming
7. Python supports all major databases
8. Python is dynamic typed
9. Python provides simple syntaxes
10. Python is extensible.

#1.python code to read last 'n' lines from a file

```
file1=input("Enter the file path:")
n=int(input("Enter the value of n:"))
```

```
with open(file1) as f1:
    print("Last",n,"lines are:")
    for line in (f1.readlines)[-n:]:
        print(line)
```

```
#2.python code to read first 'n' lines from a file
```

```
file1=input("Enter the file path:")
n=int(input("Enter the value of n:"))

with open(file1) as f1:
    print("First",n,"lines are:")

    for line in (f1.readlines()[ :n]):
        print(line)
```

```
#2.Reading specific lines from a file-->i.e 3rd and 5th lines
```

```
file1=input("Enter the file path:")

with open(file1) as f1:
    for line in (f1.readlines()[2:5:2]):
        print(line)
```

===== 15 Reading data from one file and writing to other file =====

filename: demo1.txt:

Python features:

1. powerfull Standard Library:      Introduced many builtin functionalities.
2. Wide variety of modules :      python supports nearly 89,300 modules  
Seperate module for Networking  
Seperate module for Testing  
Seperate module for Data Analytics  
Seperate module for GUI/Animations  
Seperate module for OS etc....so on...
3. portable: can run on any h/w platforms.
4. Interpreted : Python is not compiled,it is interpretation language like PERL and PHP
5. Object-Oriented :Supports OOPs features
6. Extendable : Adding various types of modules.
7. Database Support: Provides interface to all major databases.
8. simple : Easy to develop applications in Python

- 9.GUI Programming: supports GUI applications
10. Interactive mode : supports interactive mode for testing and debugging
11. Supports dynamic datatypes.
- 12.can be easily integrated with other languages like C,C++,java,CORBA etc..

filename: demo2.txt:

Python features:

1. powerfull Standard Library:      Introduced many builtin functionalities.
2. Wide variety of modules    :    python supports nearly 89,300 modules  
Seperate module for Networking  
Seperate module for Testing  
Seperate module for Data Analytics  
Seperate module for GUI/Animations  
Seperate module for OS etc....so on...
3. portable: can run on any h/w platforms.
4. Interpreted : Python is not compiled,it is interpretation language like PERL and PHP
5. Object-Oriented :Supports OOPs features
6. Extendable : Adding various types of modules.
7. Database Support: Provides interface to all major databases.
8. simple : Easy to develop applications in Python
- 9.GUI Programming: supports GUI applications
10. Interactive mode : supports interactive mode for testing and debugging
11. Supports dynamic datatypes.
- 12.can be easily integrated with other languages like C,C++,java,CORBA etc..

```
#4.Python code to read data from one file and writing that data to other file
```

```
""""
```

```
#l-method :using readlines()
```

```
f1=open("demo1.txt","r")
```

```

x=f1.readlines()
f1.close()

f2=open("demo2.txt","w")
for p in x:
    f2.write(p)
f2.close()

f3=open("demo2.txt","r")
print(f3.read())
f3.close()
"""

#II-method ---->use read() method

f=open("demo1.txt","r")
x=f.read()
f.close()

f1=open("demo2.txt","w")
f1.write(x)
#or
"""
for p in x:
    f1.write(p)
"""

f1.close()

f3=open("demo2.txt","r")
print(f3.read())
f3.close()

```

**===== 16 Reading multiple files and writing to a single file =====**

file name: emp1.txt:

101,Ajay,50000,m,11  
 102,Rahul,60000,m,12  
 103,James,30000,m,11  
 104,Amar,40000,m,12  
 105,Rohith,50000,m,13  
 106,Gill,60000,m,12  
 107,Ajith,70000,m,13  
 108,Alia,80000,f,12  
 109,Julia,90000,f,13

file name: emp2.txt

```
201,Ajay,50000,m,11  
202,Rahul,60000,m,12  
203,James,30000,m,11
```

file name: emps.txt:

```
101,Ajay,50000,m,11  
102,Rahul,60000,m,12  
103,James,30000,m,11  
104,Amar,40000,m,12  
105,Rohith,50000,m,13  
106,Gill,60000,m,12  
107,Ajith,70000,m,13  
108,Alia,80000,f,12  
109,Julia,90000,f,13  
201,Ajay,50000,m,11  
202,Rahul,60000,m,12  
203,James,30000,m,11
```

#REading 2 .txt files and writing to a single file

```
f1=open("emps1.txt","r")  
x=f1.read()  
f1.close()
```

```
f2=open("emps2.txt","r")  
y=f2.read()  
f2.close()
```

```
f3=open("emps.txt","w")  
f3.write(x)  
f3.write("\n")  
f3.write(y)  
f3.close()
```

```
f4=open("emps.txt","r")  
print(f4.read())  
f4.close()
```

===== 17 overwrite particular line=====

file name: demo1.txt:

1. Python is Simple
2. Python is user-friendly
3. Python supports interactive mode

hello hyderabad  
5. Python provides simple syntaxes  
6. Python supports object-oriented programming  
7. Python supports all major databases  
8. Python is dynamic typed  
9. Python provides simple syntaxes  
10. Python is extensible.

```
#overwriting the 4th line
f1=open("demo1.txt",mode="r")
x=f1.readlines()
x[3]="hello hyderabad\n"
f1.close()

f2=open("demo1.txt",mode="w+")
for p in x:
    f2.write(p)
f2.seek(0)
print(f2.read())
f2.close()
```

```
#reading the 4th line without readlines()
f1=open("demo1.txt",mode="r+")

f1.readline()
f1.readline()
f1.readline()
print(f1.readline())

f1.close()
```

===== 18 Changing the delimiter of a file =====

file name: emps1.txt:  
101 Ajay 50000 m 11  
102 Rahul 60000 m 12  
103 James 30000 m 11  
104 Amar 40000 m 12  
105 Rohith 50000 m 13  
106 Gill 60000 m 12  
107 Ajith 70000 m 13  
108 Alia 80000 f 12  
109 Julia 90000 f 13

```
f=open("emps1.txt",mode="r")
x=f.read()
f.close()

f=open("emps1.txt",mode="w+")

y=x.replace(","," ")
f.write(y)
f.seek(0)
print(f.read())
f.close()
```

===== 19 read specific lines and write to another file =====

file name: emps1.txt:

```
101,Ajay,50000,m,11
102,Rahul,60000,m,12
103,James,30000,m,11
104,Amar,40000,m,12
105,Rohith,50000,m,13
106,Gill,60000,m,12
107,Ajith,70000,m,13
108,Alia,80000,f,12
109,Julia,90000,f,13
```

file name: emps2.txt:

```
101,Ajay,50000,m,11
102,Rahul,60000,m,12
103,James,30000,m,11
```

#Read top3 emps and write them to seperate file

```
f=open("emps1.txt",mode="r")
x=f.readlines()
f.close()

f=open("emps2.txt",mode="w+")

y=x[0:3]
```

```
for p in y:  
    f.write(p)
```

```
f.seek(0)  
print(f.read())  
f.close()
```

===== 20 reading a file and adding some data and writing to another file =====

file name: emps1.txt:

```
101,Ajay,50000,m,11  
102,Rahul,60000,m,12  
103,James,30000,m,11  
104,Amar,40000,m,12  
105,Rohith,50000,m,13  
106,Gill,60000,m,12  
107,Ajith,70000,m,13  
108,Alia,80000,f,12  
109,Julia,90000,f,13
```

file name: emps2.txt:

```
101,Ajay,50000,m,11  
102,Rahul,60000,m,12  
103,James,30000,m,11  
104,Amar,40000,m,12  
105,Rohith,50000,m,13  
106,Gill,60000,m,12  
107,Ajith,70000,m,13  
108,Alia,80000,f,12  
109,Julia,90000,f,13  
110,Ajith,95000,m,11  
111,Amith,85000,m,12
```

#Read entire file and add some data and write them to seperate file

```
f=open("emps1.txt",mode="r")
```

```
x=f.readlines()
```

```
f.close()
```

```
f=open("emps2.txt",mode="w+")
```

```
x.append("\n110,Ajith,95000,m,11")
x.append("\n111,Amith,85000,m,12")

for p in x:
    f.write(p)

f.seek(0)
print(f.read())
f.close()
```

### **Chapter 03: Regular Expression Total notes=====**

#### **#Regular Expressions:**

Regular Expressions are used for

- 1)Extracting the required data from the given data or a string
- 2)To perform data validations
- 3)To develop url patterns in the web application.

ex:whenever we give request for a web appln, it searches for Url pattern  
if 1000 web pages ----->1000 url patterns are required,  
so it takes lot of time for searching  
so use R.E,  
for 1000webpages----->only 100 urls are enough  
for 100webpages----->only one url is enough  
use R.E,so it takes less time for searching

---

In R.E's,we use some special characters to define the patterns,  
after defining the patterns,we can extract that pattern matching  
data from the given data,by using pre-defined functions of RE module.

-re is a in-built module of python.

#The following are some special characters used to define the patterns.

1)\* : It matches with zero or more occurrences of the preceding character

ex: ab\*c

here the preceding character is b, b can occur for 0 or more times  
the above RE matches with the following strings

ac  
abc  
aBc---->Invalid  
abbc  
abbbc  
abbbb  
abc----->valid  
Abc----->invalid  
abbbd---->invalid  
cbbbb---->invalid  
abbbfc---->invalid  
abbbbb---->Invalid  
abbcabbc-->Invalid

2) a\*bc

bc  
abc  
aabc  
aaabc  
aaaabc  
aaaaac---->Invalid  
aaaabbc--->Invalid

3) a\*b\*o

o ----->valid  
ao ----->valid  
bo ----->valid  
abo----->valid  
ababo---->Invalid  
abbo----->valid  
abbboo---->Invalid  
aaboo----->valid  
bbo----->Valid  
bbb----->Invalid  
aabbo--->valid

---

2)+ : It matches with one or more occurrences of the preceding character.

ex: ab+c

here preceding character is b.

The above RE matches with the following

abc  
abbc  
abbbc  
abbbbc  
abbbbb  
ac---->invalid

ex:

b\*----->,b,bb,bbb,bbbb  
b+----->b,bb,bbb,bbbb

ex:2

a+b+c  
bc---->Invalid  
abc---->valid  
aabc---->valid  
aabbc--->Valid  
aabbcc-->Invalid  
aaaabc-->valid

---

3)? : It matches with zero or one occurrence of preceding character

ex: ab?c

the RE matches with the following

ac  
abc  
abbc---->invalid

ex:2

pea?rl  
matches with  
perl  
pearl  
pearl-->invalid

ex:3

goo?d  
god ---->valid  
good---->valid  
gd ---->Invalid

ex:4

colou?r

matches with  
color  
colour

#we can also use multiple patterns  
go\*d?

The strings supported are

g ----->valid  
gd----->valid  
go----->valid  
god----->valid  
good----->valid  
godd----->Invalid  
goood----->Valid

4). : It matches with any one single character

ex: a.c

matches with  
agc  
a5c  
a\$c  
a c  
adc  
aac  
aAc  
aac  
acc  
abc  
a+c  
a@c

#Here all the strings supported by the above pattern will be of length 3

#examples:

**ex:1----> a\* ----->various strings supported are----->{ ,a,aa,aaa,aaaa,aaaaa,aaaaaaaa}**

ex:2----> a+-----> {a,aa,aaa,aaaa,.....}

ex:3---> ba?d -----> {bd,bad}

ex:4----> b.d----->{bad,bed,bid,bud,b1d,b[+]d,b[\*]d,b d,b-d,.....}

ex:6----> a.\* ---->{a,ant, and, atm, all, animal, aunty, air, arun, azeez, alphabet.....}

ex:7---->.\*d----->{d,dd,ddd,bad,and,bold,bird,behind,dravid,card,dad,sound}  
.....d

---

5)[ ]:It matches with any single character in the given list,

ex: b[aeiou8]d  
all these character(a,e,i,o,u,8) can be present b/w b and d such as  
here the length should be only 3  
bad  
bed  
bid  
bod  
bud  
b8d  
b7d---->invalid  
bpd---->invalid

ex:2  
b[aeiour8nlhd]\*d ----->bd,bid,bed,bud,bad,bird,bend,bond,breed,board  
bond,bind,bored,board,buried,blend,behind  
bind ,blended,branded,bald,

ba\*d --->bad,baad,baaad  
# b[aeiour8nlhd]\*d--->b[aeiour8nlhd]d, --->b[aeiour8nlhd][aeiour8nlhd]d  
  
bend,  
baaad---->valid  
baeid ---->valid  
beeaaaaad-->valid  
bbaeid--->invalid  
bd ----->valid

---

6)[^]:It matches with any single character other than in the given list

ex: b[^aeiou]d  
other than these characters(a,e,i,o,u),any character can be  
b/w b and d such as  
b8d--->valid  
bpd--->valid  
bad--->invalid  
bed--->invalid  
bid--->invalid  
bod--->invalid  
bud--->invalid

---

7)[-]:It matches with any single character in the given range

ex: m[a-e]d

the range is from a to e

matches with the following

mad----->valid

med----->valid

mod----->invalid

ex:2

[0-9]----->any single digit

[a-z]----->any one lowercase alphabet

[A-Z]----->any one uppercase alphabet

[a-zA-Z]----->any one alphabet

[a-zA-Z]\*----->any no of alphabets----->{"hyd","Hyderabad","Blake","James"}

[a-zA-Z0-9\_]-->any one alpha numeric,underscore also allowed

[a-zA-Z0-9\_]\* ex: Hyd

Aug\_15----->valid

covid-19----->Invalid

covid19 ----->valid

Dec 31st----->invalid

[^0-9]----->any single non digit.

[^a-z]----->any one non lowercase alphabet

[^A-Z]----->any one non uppercase alphabet

[^a-zA-Z]----->any one non alphabet

[^a-zA-Z0-9\_]-->any one non alphanumeric (special characters)

---

8) ():match with any one string in the list.

ex:(java|C|python)

---

9){m}: It matches with exact occurrence of preceding character.

ex: ab{3}c

matches with exactly 3 occurrence of b such as

abbbc--->valid

abbc--->invalid

abbbbbbbbc-->invalid

bbb ----->invalid

R{3} ----->RRR

cp{2}---->cpp

OT{2}---->OTT

C{2}---->C++

10)^---->xoR--->start of the line

ex: ^perl

**^[abc]----->anil,blake,charan**

<sup>^</sup>[<sup>^</sup>abc]----->dravid,Rahul

11)\$ :end of the line

ex: perl\$

[0-9]\$ ----->A1,

world\$ ----->"Hello world","Python world",

12)\d or [0-9] --->any single digit

## Representing 4 digit number:

[0-9][0-9][0-9][0-9]

(or)

[0-9]{4} #means the preceding character [0-9] occurs for 4 times

(or)

\d\d\d\d

(or)

\d{4} #preceding character occurs for 4 times

#accepting any number of any number of digits---->

\d+ or [0-9]+

13)\D or [^0-9]---->any single non digit

14)\w or [a-zA-Z0-9\_]---->any one alphanumeric -->{a,B,7,\_x}

\w\* [a-zA-Z0-9\_]\* -----here multiple occurences----->{"MAY","May24","May\_24}

15) \W or [^a-zA-Z0-9\_] ----> any non alphanumeric --> only special character.

16)\s----> ' ','\t','\n' i.e space,tabspace and newline

examples:

1. "JAN 15,MARCH 05,April 16,May 23,52,101"

here JAN,MARCH,APRIL all are strings(non-numeric)

15,05,16 are numeric means--->[0-9][0-9] or [0-9]{2} or \d{2} (or)\d\d before numeric space--->''

so [a-zA-Z]\*---->means 0 or more occurrences

but i dont want 0 occurence

so finally

[a-zA-Z]+ '[0-9][0-9] | [a-zA-Z]+[0-9][0-9]

or

[a-zA-Z]+ '?'?[0-9][0-9]

ex: JAN space numeric  
re is a pre-defined module in python, which has functions or methods for Regular expressions

---

```
#re module defines many functions
#RegEx Functions
''' The re module offers a set of functions that allows us to search a string for a
match:

Function      Description
.findall       Returns a list containing of all the matches
.finditer      Returns a iterator object
.search        Returns a Match object if there is a match anywhere in the string
.sub           Replaces one or many matches with a string
.match         for matchings   ''
#-----
#findall():
#ex:1
#1)[]---->A set of characters
#ex:[a-m]

import re

str1 = "Python supports Dynamic datatypes"

#Task: Find all lower case characters  between "a" and "m" from the above string
#list=re.findall("pattern","search string") #syntax for.findall
x = re.findall("[a-m]",str1 ) #findall() returns list containing all matches

print(x,type(x))

print("No of matched characters=",len(x))

# 'a' occurred for how many times??
print(x.count("a"))
print(str1.count("a"))

print("\n")
#-----
#ex:2
#2) \d ---->any one digit
```

```

import re

#Find all digital characters from the below string

str1 = "Kohli scored 92 runs today out of 114 balls"

#Find only the numeric values(one digit,2 digit,3digit....) from the above string

x=re.findall("\d+",str1)
print(x,type(x),len(x))

#Extract only one digit no's

x = re.findall("\d", str1)      #for all 1 digit no's
print(x)

x = re.findall("\d{2}", str1)    #for all 2 digit no's
print(x)

#or
x = re.findall("\d\d", str1)    #for all 2 digit no's
print(x)
#or
x = re.findall("[0-9][0-9]", str1)    #for all 2 digit no's
print(x)
#or
x = re.findall("[0-9]{2}", str1)    #for all 2 digit no's
print(x)

x = re.findall("\d{3}", str1)      #for all 3 digits
print(x)
print("\n")

x = re.findall("\d+", str1)      #for all digits
print(x)
print("\n")

#Extract only 2 digit numbers (ex:92)

x = re.findall("\d+", str1)      #for all digits
print(x)
print("\n")

y=[]  #append only 2 digit numbers into y
for p in x:
    if(len(p)==2):

```

```

        y.append(p)
print(y)

#-----
#ex:2
str1="101.45 102 30 5 57.35"

x = re.findall("\d+", str1)
print(x)

#ex:3
str1="101.45,102,30,5,57.35"

x = re.findall("\d+", str1)
print(x)

#Extract only float values
x=re.findall("\d+[.]\d+",str1)
print(x)

x=re.findall("\d+[.]\d+|\d+",str1)
print(x)
x=re.findall("\d+[.]?\d*",str1)# here dot occurred for 1 time means--->float value
                                #here dot occurred for 0 times means--->int value
print(x)
"""

x=re.findall("\d+|\d+[.]\d+",str1)
print(x)
"""

#I want only the integer values but not floats
x=re.findall("\d+[.]\d+|\d+",str1)
print(x)

y=[]
for p in x:
    if('.' not in p):
        y.append(p)
print(y)

#-----
#ex:1
"""
1)6.3 =====>\d[.]\d (or) [0-9][.][0-9]

```

2) 6.72====> \d[.] \d\d (or) \d[.] \d{2} (or) [0-9][.][0-9][0-9] (or) [0-9][.][0-9]{2}

3) 64.786==> \d\d[.] \d\d\d (or) \d{2}[.] \d{3} (or) [0-9]{2}[.] [0-9]{3}  
for all the above patterns the generalized pattern is==> \d+[.] \d+

The above patterns wont extract a integer like 350

so for integers the pattern is----->\d+

for integers and floats the pattern is----->\d+[.] \d+ | \d+

"

```
#-----
#ex:3
#3)^ ----->starts with
import re

str = "hello world"

#Check if the string starts with 'hello':

x = re.findall("^hello", str)
print(x,type(x)) #returns a list
if(x):
    print("Yes, the string starts with 'hello'")
else:
    print("No match")
print("\n")

#-----
#ex:4
#4)\$ ----->ends with
import re

str = "hello world"

#Check if the string ends with 'world':

x = re.findall("world$", str)
print(x,type(x))
if(x):
    print("Yes, the string ends with 'world'")
else:
    print("No match")
print("\n")

#-----
#ex:5
#5)* -----> 0 or more occurences of preceeding character
```

```
import re

str = "The rain in Spain falls mainly in the plain!...."

#Check if the string contains "ai" followed by 0 or more "x" characters:

x = re.findall("aix*", str)
print(x)
#Check if the string contains "ai" followed by 0 or more "n" characters:

x = re.findall("ain*", str)
print(x)
#-----
#ex:6
#6)+ -----> 1 or more occurences
import re

str = "The rain in Spain falls mainly in the plain!..."

#Check if the string contains "ai" followed by 1 or more "x" characters:

x = re.findall("aix+", str)
print(x)

#Extract the words containing "ai" within it.

x = re.findall("[a-zA-Z]+ai[a-zA-Z]+", str)
print(x)

#or
x = re.findall("\w+ai\w+", str)
print(x)

#ex:2
str = "The rain in Spain falls mainly in the plain!... air "

x = re.findall("[a-zA-Z]*ai[a-zA-Z]+", str)
print(x)
#or
x = re.findall("\w*ai\w+", str)
print(x)
```

```

#Extract the words which consists "in" within it

x = re.findall("[a-zA-Z]*in[a-zA-Z]*", str)
print(x)

#or
x = re.findall("\w*in\w*", str)
print(x)

#ex:3
names="Ajay,Rohit,praveen,Blake,Puja,James,Pushpa,Dhoni,supriya,Python3"
#Extract only the names starting with P

x=re.findall("P[a-zA-Z]+",names)
print(x)
#or
x=re.findall("P\w+",names)
print(x)
#Extract-->irrespective of the case

x=re.findall("P\w+|p\w+",names)
print(x)
#or
x=re.findall("[Pp]\w+",names)
print(x)

#check and extract whether the name Dhoni is available

x=re.findall("Dhoni",names)
print(x)
if(x):
    print("match")
else:
    print("No match")

#ex:1)Write a pattern which can extract any 3digit number
#    2)write a pattern which can extract any 3 digit number starting with 3
#1.
y="102,203,304,501,30,307,92,920,354,982"
x=re.findall("\d\d\d",y)
print(x)
#2.
x=re.findall("3\d\d",y)

```

```

print(x)

#ex:4
ids="401,501,402,502,403,503,101,202,107,525,322,540"

#Extract the id's of CSE students only    CSE-5 ,ECE-4, EEE-2 ,MECH-3 ,CIVIL-1

x=re.findall("5[0-9]{2}",ids)
print(x)

#or
x=re.findall("5[0-9][0-9]",ids)
print(x)

#or
x=re.findall("5\d\d",ids)
print(x)

#or
x=re.findall("5\d{2}",ids)
print(x)

#Task : Extract only ECE and CSE students

x=re.findall("[4-5][0-9]{2}",ids)
print(x)
#or
x=re.findall("[4|5][0-9]{2}",ids)
print(x)
#or
x=re.findall("[45][0-9]{2}",ids)
print(x)
#or
x=re.findall("[45]\d{2}",ids)
print(x)
#or
x=re.findall("4\d{2}|5\d{2}",ids)
print(x)

#Task: Extract all the ids

x=re.findall("\d\d\d",ids) #or x=re.findall("\d{3}",ids) #or x=re.findall("[0-9]{3}",ids)
print(x)
#or

```

```
x=re.findall("[1-5][0-9]{2}",ids)
print(x)
#or
x=re.findall("[1-5][0-9][0-9]",ids)
print(x)
#or
x=re.findall("[1-5]\d{2}",ids)
print(x)
#or
x=re.findall("[12345]\d{2}",ids)
print(x)

#-----
#ex:7
#7) {m}:Exactly the specified number of occurrences
```

```
import re

str = "The rain in Spain falls mainly in the plain!"

#Check if the string contains "a" followed by exactly two "l" characters:
```

```
x = re.findall("al{2}",str)
print(x)
#or
x = re.findall("all", str)
print(x)
```

```
#Extract the word/words containing all
```

```
x = re.findall("\w+all\w+", str)
print(x)
#-----
#8) |
```

```
str1="Python is easier than C Language"
#Check if the string contains either "Python" or "Java":
```

```
x = re.findall("Python|Java", str1)

print(x)

if(x):
    print("Yes, there is at least one match!")
else:
    print("No match")
#-----
```

```
#Extract the names starting with a/A

y="ajith,Amar,Rohith,Kohli,Sameer,Anil,amar"

x=re.findall("A[a-z]+|a[a-z]+",y)
print(x)

x=re.findall("[Aa][a-z]+",y)
print(x)

x=re.findall("[A | a][a-z]+",y)
print(x)
```

## #Examples

```
import re

#ex1: Extract only years from the following data
#Extract only years and construct a list from it.
```

```
data=""
1st worldcup held in the year 1975
2nd worldcup held in the year 1979
3rd worldcup held in the year 1983
4th worldcup held in the year 1987
5th worldcup held in the year 1992
6th worldcup held in the year 1996
7th worldcup held in the year 1999
8th worldcup held in the year 2003
last elections held in the year 2024
previous elections held in the year 2019
""
```

```
x=re.findall("\d\d\d\d",data)
print(x)
#or
x=re.findall("\d{4}",data)
print(x)
#or
x=re.findall("[0-9][0-9][0-9][0-9]",data)
print(x)
#or
x=re.findall("[0-9]{4}",data)
```

```

print(x)
#Extract only the years in which cricket world cups are held

x=re.findall("[a-zA-Z0-9 ]+worldcup[a-zA-Z0-9 ]+([0-9]{4})",data)
print(x)
#or
x=re.findall("[\w ]+worldcup[\w ]+([0-9]{4})",data)
print(x)
#Note: in the above example, 1st it checks for the pattern and extracts only the
#one which is within braces

#Ex:2 I want all 1 digit,2 digit, 3 digit ,4digit no's (only numericals) from the above string

x=re.findall("\d+",data)
print(x)

#Ex:3 A pattern which can extract numericals and alphabets

str="a,A,_9,$,%"

x=re.findall("\w",str)
print(x)

#or I doesnt want underscore

x=re.findall("[a-zA-Z0-9]",str)
print(x)

#Ex:4 A pattern which can extract only special symbols

x=re.findall("\W",str)
print(x)

#ex:5 only special symbols excluding comma

x=re.findall("[^A-Za-z0-9_,]",str)
print(x)
#or
x=re.findall("[^\w,]",str)

```

```
print(x)
```

#ex:6 A pattern wwhich can extract all these

```
str1="oct2,Jan26,Aug15,Jan_1,August,2019"
```

```
x=re.findall("\w+",str1)
print(x)
#or
x=re.findall("[a-zA-Z0-9_]+",str1)
print(x)
```

#ex:7 A pattern wwhich can extract all these

```
str1="oct2,Jan26,Aug15,Jan_1,August,2019,45000$,720"
```

```
x=re.findall("[a-zA-Z0-9_\\$]+",str1)
print(x)
#or
x=re.findall("[\\w\\$]+",str1)
print(x)
```

#ex:8 A pattern which can extract only special symbols

```
str1="oct2,Jan26,Aug15,Jan_1,August,2019,45000$,720"
```

```
x=re.findall("\W",str1)
print(x)
```

#ex:9 A pattern which can extract only \$ excluding comma  
str1="oct2,Jan26,Aug15,Jan\_1,August,2019,45000\$,720"

```
x=re.findall("\[$]",str1)
print(x)
#or
x=re.findall("[^A-Za-z0-9_]",str1)
print(x)
#or
x=re.findall("[^\\w,]",str1)
print(x)
```

#ex:10 A pattern which can extract \$ along with the numericals attached to it

```

str1="oct2,Jan26,Aug15,Jan_1,August,2019,45000$,$720"

x=re.findall("\d*[$]\d*",str1) #or re.findall("\d+[$]|[$]\d+",str1)
print(x)

#ex:11 A pattern which can extract any number,lowercase,uppercase,special characters

str1="7,a,A,bb,_,$,+,*,++,hyderabad,me@gmail.com"

x=re.findall(".",str1)
print(x)

```

#ex:12

```

#Extract only height values from the following data and compute the max height value
data="" Blake=6.3,
Miller=5.8,
James=5.11,
John=5,
Amar=6.5"""


```

```

x=re.findall("\d[.]\d+|\d+",data)
print(x)
#or
x=re.findall("\d[.]?\d*",data)
print(x)

```

```

y=[] #empty list

for p in x:
    y.append(float(p))
print(y)
print("max height=",max(y))


```

```

#ex:13
#Extract only weight values from the following data
data="" Blake=82.57,
Miller=78.25,
James=105.90,
John=89
Amar=97.30"""


```

```

x=re.findall("\d+[.]\d+|\d+",data)
print(x)

#ex:14
#Extract only scores from the following data and compute the total score
scorecard="Rohith=45
Kohli=104
Dhoni=34
Rahul=44
pant=38
Gill=8
"
x=re.findall("\d+",scorecard)
print(x)

scores=[]

for p in x:
    scores.append(int(p))
print(scores)
print("Total Score=",sum(scores))

import re
regex="[a-zA-Z]+\d*" #, regex is a variable
#+ indicates one or more occurance of a character, followed by
#one space,followed by 1 or more occurance of a digit.
x=re.findall(regex,"June 15,August 9,Dec12,May ,oct ,Feb 2020,2021")
#findall is a fn for searching a given pattern in given string
#it takes 2 parameters--> regex patten and search string
#x is a list of matched strings
print(x)
for match in x:
    print("Full match:",match)

#Note:if-----> regex="[a-zA-Z]+\d*----->then May ,oct also will be extracted

```

```

import re
regex="[a-zA-Z]+\d+"

```

```
matches=re.findall(regex,"June 15,July 15,August 9,Dec 12,12 Feb") #12 Feb is invalid
print(matches)
for match in matches:
    print("Full match",match)
```

#finditer():Returns a iterator object which yields Match object for every match

```
#To print the start and end of the match.
import re
regex="[a-zA-Z]+\d+"
matches=re.finditer(regex,"June 15,August 9,Dec,Jan 31,Feb 21")
#finditer() returns iterator object
#findall() returns list object,here list doesnt support start() and end() methods
print(matches) #Iterator object prints only the address, it wont prints the content
for match in matches:
    print("match start index and end index:",match.start(),match.end())
```

#To capture DOB--->month and day

```
import re
regex="[a-zA-Z]+\d+"
x=re.findall(regex,"June 15th,August 9,Dec 12,Feb 22,jul ,hello ,Chennai") #jul is invalid
print(x)
for match in x:
    print(match)
print("\n")
```

#II) #To capture only month

```
regex="([a-zA-Z]+) \d+" #it extracts only the pattern within parenthesis
x=re.findall(regex,"June 15th hyd,August 9,Dec 12,Feb 22,jul ,hello,chennai") #jul is invalid
print(x)
for match in x:
    print("month:",match)
print("\n")
```

#III) #To capture only date but not month

```
regex="[a-zA-Z]+ (\d+)" #it extracts only the pattern within parenthesis
x=re.findall(regex,"June 15th,August 9,Dec 12,Feb 22,jul ,2020,2019") #jul is invalid
print(x)
for match in x:
```

```
    print("Date:",match)  
print("\n")
```

```
#ex:3  
data = "Miller DOB: June 15th,  
       Blake DOB: August 9th  
       Ajay DOB: Febrauary 21st  
       Amar DOB: march 17th"
```

#Extract only month and day

```
x=re.findall("[a-zA-Z]+\d+",data)
print(x)
#or
regex1="[a-zA-Z]+\d+"
x=re.findall(regex1,data)
print(x)
print("\n")
```

```
#ex:4  
#extract only emp names now  
data = "Miller DOB: June 15th,  
Blake DOB: August 9th  
Ajay DOB: Febrauary 21st  
Amar DOB: march 17th"
```

```
x=re.findall("[a-zA-Z]+ [a-zA-Z]+",data)
print(x)
#or
x=re.findall("[a-zA-Z]+ [A-Z]+",data)
print(x)
#or
y=re.findall("\w+ [A-Z]+",data)
print(y)
#or
x=re.findall("[a-zA-Z]+ [D]",data)
print(x)
#or
x=re.findall("[a-zA-Z]+ DOB",data)
print(x)
#or
x=re.findall("(.) DOB",data)
print(x)#
print(x)#

```

```

#ex:5
#extract only names and months

data="""Miller DOB: June 15th,
Blake DOB: August 9th,
Ajay DOB: Febrauary 21st,
Amar DOB: march 17th"""

x=re.findall("[a-zA-Z]+ ",data)
print(x)
#or
regex2="(\w+ )"
y=re.findall(regex2,data)
print(y)
#or
x=re.findall("[a-zA-Z]+ [A-Z:]+ ([a-zA-Z]+)",data)
print(x)
#or
x=re.findall("[a-zA-Z]+ DOB: ([a-zA-Z]+)",data)
print(x)

#Extract only Miller and June

x=re.findall("Miller |June",data)
print(x)

#extract only--> DOB

x=re.findall("[A-Z]+:",data)
print(x)
#-----
#zip():for merging 2 lists

subs=["maths","phy","chem"]

marks=[90,80,70]

#Now merging both the lists using zip()
x=zip(subs,marks)
print(x,type(x))

y=list(x)
print(y,type(y))

z=dict(y)
print(z,type(z))

```

```

#-----
#ex:6
#Create a dictionary from the below data by extracting names as key and months as value
#first create 2 lists-->names and months and merge those lists and convert to dictionary
#using zip() function

#extract only emp names as one list

data="""Miller DOB: June 15th,
Blake DOB: August 9th,
Ajay DOB: February 21st,
Amar DOB: march 17th"""

#Extract names
regex2="([a-zA-z]+) DOB"
enames=re.findall(regex2,data)
print(enames)

#extract only months
regex2=": ([a-zA-z]+)"
months=re.findall(regex2,data)
print(months)

#Merging 2 lists using zip() function
x=zip(enames,months)
print(x,type(x))

y=list(x)
print(y,type(y))

z=dict(y)
print(z,type(z))

#sub() :for substitutions
#To print date and month
import re
regex="([a-zA-Z]+) (\d+)" #extracts month and also date bcoz both r in parenthesis
print(re.sub(regex,r"\2 of \1","June 15,August 9,Dec 22,Jan 31,Feb 21"))
#\2 of \1, means 15 of June,means 2nd pattern extracted first and then extracts 1st pattern
#To print each in separate line

#help(re)

```

```
import re
regex="([a-zA-Z]+) (\w+)"
x=re.sub(regex,r"\2 is capital of \1","Telangana Hyderabad,Ap Amaravathi,Karnataka
Bengluru,maharashtra Mumbai")#x stores o/p of sub---->15 of June,....
print(x)
```

```
import re
regex="([a-zA-Z]+) (\w+) (\d+)"
x=re.sub(regex,r"\2 capital of \1 has covid cases of \3","Telangana Hyderabad 15000,Ap Amaravathi
25000,Karnataka Bengluru 30000,maharashtra Mumbai 40000")#x stores o/p of sub---->15 of June,....
print(x)
print("\n")
```

```
#ex:2
x=re.sub(regex,r"\1 has covid cases of \3","Telangana Hyderabad 15000,Ap Amaravathi
25000,Karnataka Bengluru 30000,maharashtra Mumbai 40000")#x stores o/p of sub---->15 of June,...
print(x)
```

```
#Return the domain type of given email-ids
import re
regex='@(\w+)[.]\w+'
result=re.findall(regex,'mahendra@gmail.com, osmania@ac.in, techm@online.com,
abc@rest.biz,rajesh@yahoo.co.in')

print(result)
```

```
#Return the domain type including .com or .in etc of given email-ids
import re
regex='@\w+[.][\w.]+'
result=re.findall(regex,'Techm@gmail.com, osmania@ac.in, naresh@online.com,
abc@rest.biz,naresh@yahoo.co.in')
print(result)
```

```
#re.match():
#Validate a phone number: phone number must be of 10 digits and starts with 6 or 7 or 8 or 9
import re
x=['9983764321','7384963897','5376412769','938476182']
for val in x:
    if(re.match('[6-9]{1}[0-9]{9}',val) and len(val) == 10 ):
        print('Valid')
    else:
        print('Invalid')
```

```
#search()
import re

#Check if the string starts with "python" and ends with "java":

txt = "python is easier and simpler than Java"
x = re.search("^python.*Java$", txt)#. indicates any character,* indicates multiple occurrences
#print(x)
if (x):
    print("Matching")
else:
    print("No match")
```

```
#re.split()
#The re.split method splits the string where there is a match and returns a list of strings
#where the splits have occurred.

import re

string = 'Twelve:12 Eighty nine:89.'
pattern = '\d+'

result1 = re.split(pattern, string)
print(result1)
# Output: ['Twelve:', ' Eighty nine:', '.']
```

```

#-----

#ex:2
string = 'Twelve:12 Eighty nine:89.'
pattern = ''

result1 = re.split(pattern, string)
print(result1)
#o/p:
['Twelve:12', 'Eighty', 'nine:89.']
#-----

#ex:3
string = 'Twelve:12 Eighty nine:89.'
pattern = ':'

result1 = re.split(pattern, string)
print(result1)
#o/p:
['Twelve', '12 Eighty nine', '89.']
#-----

#ex:4
string = 'Good Morning Hyderabad'
pattern = '\d'

result1 = re.split(pattern, string)
print(result1)
#o/p:
['Good Morning Hyderabad']
#If the pattern is not found, re.split() returns a list containing the original string.

```

#You can pass maxsplit argument to the re.split() method.  
#It's the maximum number of splits that will occur.

```

import re

string = 'Twelve:12 Eighty nine:89 Nine:9.'
pattern = '\d'

# maxsplit = 1
# split only at the first occurrence
result = re.split(pattern, string, 1)
print(result)

```

```
# Output: ['Twelve:', ' Eighty nine:89 Nine:9.']
#-----
#ex:2
string = 'Twelve:12 Eighty nine:89 Nine:9.'
pattern = '\d+'

# maxsplit = 2
result = re.split(pattern, string, 2)
print(result)
# output :['Twelve:', ' Eighty nine:', ' Nine:9.']}
```

## **Chapter 04: Exception handling Total notes=====**

### **Exception Handling:**

Generally, there are 2 types of errors in python

- 1.Syntax errors
- 2.Run-time errors

1)Syntax errors: The errors that occur due to wrong syntaxes

ex: wrong space indentation  
      missing colon

ex:

x=[10,20,30,40,50]

```
for p in x #syntax error---->missing colon
print(p)   #syntax error---->wrong indentation
```

Python interpreter checks for syntax errors and then converts source code to byte code(.cpy) and then converts this byte code to machine code and executes.

If there are syntax errors,then .cpy file wont be generated,  
If .cpy file is not generated then the python program wont be executed.

---

2)Run-time errors:    Errors that occur at run-time,i.e during the execution of the program.

ex:Trying to access an element which is not present ---->we get Runtime Error  
    Trying to remove an element which is not present---->we get Runtime error

```
ex-1:  
x=[10,20,30,40,50]  
trying to access 10th element---Runtime error  
print(x[10])---Runtime error--->IndexError
```

ex-2: Trying to remove an element which is not present----> gives runtime error  
ex: x.remove(90)---->Runtime error---->ValueError

#Run-time errors:

```
x=[10,20,30,40,50]  
  
#print(x[9])           #IndexError  
  
#x.remove(90)          #ValueError  
  
#print(y)              #NameError  
  
#y=list(10)            #TypeError  
  
#z=list("hello","world") #TypeError  
  
  
y=(10,20,30,40,50)  
  
#y.append(60)  ----->AttributeError  
  
#z.append(70)  ----->AttributeError  
  
#print("python"+3)      #TypeError  
  
#f=open('sample5.txt')    #FileNotFoundException  
  
x=10  
y=0  
#z=x/y                  #ZeroDivisionError  
  
  
  
#x=int(input("Enter value of x:")) --->"hello" #ValueError
```

whenever we get run-time error, program execution is terminated abnormally

#Abnormal Termination: Termination of the program in middle of the execution without  
executing upto the last stmt

for every run time error,corresponing pre-defined python classes are available, these classes are called as Exception classes.

Exceptions are the classes,which contain run-time error representations.

```
#we can see all the pre-defined exception classes within a module called as __builtins__
print(dir(__builtins__))
when ever we get run-time error,run-time error representation class object
will be created automatically.
whenever this object is created,we say that exception is raised, we need to
handle it with a code,otherwise abnormal termination.
```

2types of exceptions:

Pre-defined Exceptions(Built-in Exceptions)----->Raised automatically and we need  
to handle.

User-defined Exceptions----->We need to raise and we need to handle.

reasons for run-time errors:

- 1.Entering invalid input
- 2.Due to invalid code
- 3.memory related issues
- 4.Hardware related problems.

```
x=int(input("Enter First NO:"))
y=int(input("Enter Second NO:"))
z=x/y
print(z)
```

#ZeroDivisionError : if 2nd no is entered as 0

```
'''if x=10,y=3 ,then no error,
if x=10,y=0, then runtime error,(Zerodivision error),predefined exception class '''
```

#Exception Handling:

The process of identifying raised exception object and handling it by assigning that exception to corresponding run-time error representation class is known as Exception handling.

we can implement exception handling in python using try & except block.

we need to follow indentation for both try and except blocks.

1)try block:

syntax:

try:

.....

.....

.....

..... for identifying raised exception

The Stmt which causes run-time errors are placed within try block.

ex:

```
x=int(input("enter value of x:"))
y=int(input("enter value of y:"))
try:
    z=x/y
    print(z)
```

If exception is raised in 1st stmt of try block,then immediately control will go to except block,without executing the remaining stmts of try block.

except block syntax:

except(Exception class name):

.....  
.....  
.....  
.....

except block assigns the exception object to the corresponding run-time error representation class.

we can also display user-friendly error messages in except block.

ex:

```
except(ZeroDivisionError):#assigning exception object to corresponding run-time error representation
```

```

class
    print("second No cannot be zero")

x=int(input("Enter First No:"))
y=int(input("Enter Second No:"))

try:
    z=x/y
    print(z)

except(ZeroDivisionError):
    print("2nd number cannot be zero")

#To print the error msg and classname
#except Exception as e:
#    print(e)
#    print(type(e))

#Single try with multiple except blocks:

#In order to handle different exceptions by using different except blocks,
#then we go for
#single try with multiple except blocks.

#ValueError

#x=int(input("Enter First No:")) --->"Four" (we get valueerror),
#so keep these stmts also in try block.

try:
    x=int(input("Enter First No:"))    #10
    y=int(input("Enter Second No:"))   #two, if error here,remaining stmts of try block r skipped
    z=x/y
    print(z)
    #print(w)

```

```

except(ZeroDivisionError):
    print("2nd No cannot be zero")

except(ValueError):
    print("Enter Numerical values only")

except(NameError):
    print("variable is not defined")

except:                      #default except block
    print("error occurred")

#-ValueError except block can handle only one exception.

#ZeroDivisionError except block can handle only one exception

#-default except block can handle any type of exception.

#-In default except block,we can display only common user-friendly error msg
# but here we cannot display the corresponding exception related user-friendly error message.

#-In the above program, if exception is raised in the try block,then control will
#go to 1st except block,if 1st except block has handled that exception,then control
#will not go to remaining except blocks

# - If 1st except block has not handled that exception,then only control will go to
#     the next except block,still not handled,control goes to default except block.

#in try block, if error in 1st line only,then remaining stmts will be skipped in try block
# and control goes to except block, so in try block, only one exception it can handle ""

# If I write default except block first,then control won't go to the next except blocks
# So if I define it first, we get syntax error,
# So always define default except block at the last.

try:
    x=int(input("Enter First No:"))      #10
    y=int(input("Enter Second No:"))    #abc, if error here,remaining stmts of try block r skipped
    z=x/y

```

```
print(z)

except:                                #default except block
    print("error occured")

except(ZeroDivisionError):
    print("2nd No cannot be zero")

except(ValueError):
    print("Enter Numerical values only")
```

#try-except else block:if no exception in try block then only ctrl goes to  
#else block

```
#FileNotFoundException:
try:
    f=open("C:/data1/demo1.txt","r")      #r--->read mode
    print(f.read())
    f.close()

except(FileNotFoundError):
    print("File doesnt exist")

except:
    print("Error")

else:
    print("File opened Successully")
print("end")
```

#AttributeError

```
#x=(10,20,30,40,50)
#x.append(60) #AttributeError
#x[1]=25   #TypeError
```

```
try:
    x=(10,20,30,40,50)
    x.append(60)
    x[1]=25
```

```
except(AttributeError):
```

```

print("Tuple doesnt support Insertion")

except(TypeError):
    print("Tuple elements cannot be modified")

except:
    print("Error Occured!!!")

#NameError
#x
try:
    print("hello...")
    print(x)
    print("Good Morning...")
except(NameError):
    print("Variable is not defined")
except:
    print("Error")

```

```

#TypeError:
#a=5#
#b='3'
#print(a+b)

try:
    a=5
    b='3'
    print(a+b)

except(TypeError):
    print('Unsupported operation')

except:
    print("Error")

```

#finally block:

|                     | # I-Iteration   | II-Iteration(exception)     |
|---------------------|-----------------|-----------------------------|
| III-Iteration       |                 |                             |
| (Exception occurred | #(no exception) | (Exception occured&handled) |
| try:                |                 | #(zeroDivisionError)        |
| not handled)        |                 | &                           |

```

x=int(input("Enter First No:"))    #200          100          100
y=int(input("Enter Second No:"))   #100           0

two
z=x/y
print(z)

except(ZeroDivisionError):
    print("2nd No cannot be zero")

print("welcome to Python World!!!.....")#o/p :Welcome to      o/p: 2nd No cannot
o/p:Abnormal Termination
#      Pythonworld!!!.....      be zero
#welcome to Python.

```

#but in all the above 3 situations, I want to display "welcome to Python world!!!.....",  
#then use finally block

#finally block:The block of stmts which will be executed always,whether exception occurred or  
#not occured or exception handled or not handled, but these stmts within the finally block  
#will be executed always.

| # I-situation                    | II- situation                     | III-situation                 |
|----------------------------------|-----------------------------------|-------------------------------|
| # try:                           | try:                              | try:                          |
| #     fileopen--->success        | fileopen--->success               | fileopen-->success            |
| #     operations->success        | operations-->error                |                               |
| operations-->error               |                                   |                               |
| #so it wont go to except block   | ctrl goes to except block&handled | ctrl goes to except block&not |
| handled                          |                                   |                               |
| # f.close()---->will be executed | f.close()---->will be executed    | Abnormal Termination,         |
| #                                |                                   | f.close()---is                |
| not executed.                    |                                   |                               |

#so here in all the 3 situations ,keep f.close()(file closing stmt)in finally block so it executes in  
#all 3 situations

#file closing stmts(Resource releasing stmts) and database connection closing stmts are recommended  
to  
#represent by using finally block.

#Implementing try,except and finally  
try:  
 x=int(input("Enter First No:"))

```
y=int(input("Enter Second No:"))
z=x/y
print(z)

except(ZeroDivisionError):
    print("2nd No cannot be zero")

finally:      # here prints msg of finally block before terminating abnormally
    print("welcome to Python world!!!")

print("end")
```

#Try the execution for all the 3 cases ,in all 3 cases finally will be executed

#Different Exception HAndling

1)single try & single except block

```
try:
    .....
    .....
    .....
    .....
except:
    .....
    .....
    .....
    .....
```

2)single try & multiple except blocks

```
try:
    .....
    .....
    .....
    .....
except(Exceptionclass):
    .....
    .....
    .....
    .....
except:
    .....
```

```
.....  
.....  
.....  
#Note: only try block is not allowed without except block
```

### 3)try & finally block

```
try:  
.....  
.....  
.....  
.....  
finally:  
.....  
.....  
.....
```

This is valid syntax but not recommended bcoz,  
here exception is not handled,  
here finally just prints some messages

### 4)try ,except & finally blocks

```
try:  
.....  
.....  
.....  
.....  
except(Exceptionclass):  
.....  
.....  
.....  
.....  
finally:  
.....  
.....  
.....  
.....
```

### 5)try ,multiple except & finally blocks

```
try:  
.....  
.....  
.....  
.....  
except(Exceptionclass):  
.....  
.....
```

```
.....  
.....  
except:  
.....  
.....  
.....  
.....  
.....  
finally:  
.....  
.....  
.....  
.....
```

6)try ,except & else blocks

```
try:  
.....  
.....  
.....  
.....  
except(Exceptionclass):  
.....  
.....  
.....  
.....  
else:  
.....  
.....  
.....  
.....
```

without try block,can i define except block---->No

without except block,can i define try block---->yes, using finally block

#Nested Blocks:

#In try block--->I can have try,except,finally blocks

#in except block--->I can have try,except,finally blocks

#in finally block--->I can have try,except,finally blocks

```

try:
    print(10/0)
    print("in try1...")
try:
    #print(z)
    print("in try2.....")
except:
    print("in except2...")
finally:
    print("in finally2...")
except:
    print("in except1...")
try:
    print("in try3.....")
    print("Python"+3)
except:
    print("in except3...")
finally:
    print("in finally3...")
finally:
    print("in finally1...")
try:
    print("in try4.....")
    print(p)
except:
    print("in except4...")
finally:
    print("in finally4...")

```

#### #User-defined Exceptions:

The exceptions which are defined by the programmers explicitly according to their business requirements are known as userdefined exceptions.

steps to implement user-defined exceptions

- 1.defining user-defined exception class
- 2.Raising the Exception
- 3.Handling the Exception.

#### 1)Defining user-defined Exception class:

A user-defined Exception class that extends any one of the pre-defined exception class then the class is called user-defined Exception class.

```
syntax: class classname(Exception):
          pass
```

#2)Raising the Exception:  
we can raise the exceptions by using raise keyword.

syntax:

```
raise exceptionclassname
```

we can also raise the pre-defined exceptions explicitly by using raise keyword.

Creating exception class object explicitly is known as the raising the exception.

-predefined exceptions raised automatically

-user-defined exceptions ,we need to raise explicitly by using raise keyword.

#3)Handling the exception:

-we can handle the raised exception by using try and except blocks.

#we can also raise the pre-defined exceptions explicitly by using raise keyword.

#whether a person is eligible to vote or not.

'''

```
try:
    age = int(input("Enter the age:"))
    if(age<18):
        raise ValueError
    else:
        print("Eligible to Vote")
```

```
except ValueError:
    print("The age is less, not eligible to vote")
'''
```

```
=====
```

```
class abc(Exception):
    pass
```

```
try:  
    age = int(input("Enter the age:"))  
    if(age<18):  
        raise abc  
    else:  
        print("Eligible to Vote")  
  
except abc:  
    print("The age is less, not eligible to vote")
```

```
#ZeroDivisionError  
try:  
    a = int(input("Enter a:"))  
    b = int(input("Enter b:"))  
  
    if(b==0):  
        raise ZeroDivisionError  
    else:  
        print("a/b = ",a/b)  
  
except (ValueError):  
    print("Enter Numerical values only")  
  
except(ZeroDivisionError):  
    print("The value of b cannot be 0")
```

```
finally:  
    try:  
        print("a=",a)  
        print("b=",b)  
    except:  
        print("Invalid input")
```

```
#ValueError  
try:  
    marks=int(input('Enter the marks:'))  
    if(marks > 100):  
        raise ValueError  
  
except(ValueError):  
    print(marks, "is out of allowed range")
```

```

else:
    print("MARKS:",marks)

#try-except finally block
#FileNotFoundException:
try:
    f=None
    f=open("C:/data1/demo1.txt","r")
    if(f!=None):
        print("File opened successfully")
        print(f.read())
        #print(y)
    else:
        raise FileNotFoundError

except(FileNotFoundError):
    print("File doesnt exist")

except:
    print("Error occured")

finally:
    if(f!=None):
        f.close()
        print("File closed successfully")
    else:
        print("file not opened")

#Program to guess a number
class Error(Exception):    #Error is userdefined class that extends pred-defined class Exception
    """Base class for other exceptions"""
    pass
class ValueTooSmallError(Error): #ValueTooSmallError is a userdefined class that extends another
                                # user-defined class(Error) which extends pre-defined class
Exception
    """Raised when the i/p value is too small"""
    pass
class ValueTooLargeError(Error): #ValueTooLargeError is a userdefined class that extends another
                                # user-defined class(Error) which extends pre-defined class
Exception
    """Raised when the i/p value is too large"""
    pass

```

```

number=10
while True:      #infinite while loop
    try:
        x=int(input("Enter a Number:"))
        if(x<number):
            raise ValueTooSmallError
        elif(x>number):
            raise ValueTooLargeError
        break
    except(ValueTooSmallError):
        print("This value is too small,try again")

    except(ValueTooLargeError):
        print("This value is too large,try again")
print("CONGRATULATIONS!! You Guessed it Correctly...")

```

"""output:  
 Enter a Number:5  
 This value is too small,try again  
 Enter a Number:15  
 This value is too large,try again  
 Enter a Number:10  
 CONGRATULATIONS!! You Guessed it Correctly.."""

## **Chapter 05: Python Database connectivity notes=====**

#SQL :Structured-Query-Language

-It is the language which is followed by all the databases

#CRUD operations  
#C-CREATE  
#R-READ  
#U-UPDATE  
#D-DELETE

mysql> show databases; ---->displays all the available databases

#creating a new database

mysql> create database mydb81;  
Query OK, 1 row affected (0.05 sec)

```
mysql> use mydb81;
Database changed
```

```
mysql> show tables;
Empty set (0.05 sec)
```

```
#=====  
#Creating a table--->emp
```

```
mysql> create table emp(eid int,ename varchar(20),sal int,sex varchar(6),dno int);
```

```
mysql> insert into emp  
values(101,'Miller',10000,'m',11),(102,'Blake',20000,'m',12),(103,'Sony',30000,'f',11),(104,'Geeta',40000,  
'f',12),(105,'James',50000,'m',13);
```

```
mysql> select * from emp;  
+----+-----+-----+-----+  
| eid | ename | sal | sex | dno |  
+----+-----+-----+-----+  
| 101 | Miller | 10000 | m | 11 |  
| 102 | Blake | 20000 | m | 12 |  
| 103 | Sony | 30000 | f | 11 |  
| 104 | Geeta | 40000 | f | 12 |  
| 105 | James | 50000 | m | 13 |  
+----+-----+-----+-----+
```

```
#Retrieving selected columns
```

```
mysql> select ename,sal from emp;  
+-----+-----+  
| ename | sal |  
+-----+-----+  
| Miller | 10000 |  
| Blake | 20000 |  
| Sony | 30000 |  
| Geeta | 40000 |  
| James | 50000 |  
+-----+-----+
```

```
#=====
```

```
#Update operations
```

```
#update the salary of blake by 7000
```

```
mysql> update emp set sal=sal+7000 where ename="Blake";
```

```
mysql> select * from emp;
```

```
+----+-----+-----+-----+  
| eid | ename | sal | sex | dno |  
+----+-----+-----+-----+
```

| eid | ename  | sal   | sex | dno |
|-----|--------|-------|-----|-----|
| 101 | Miller | 10000 | m   | 11  |
| 102 | Blake  | 27000 | m   | 12  |
| 103 | Sony   | 30000 | f   | 11  |
| 104 | Geeta  | 40000 | f   | 12  |
| 105 | James  | 50000 | m   | 13  |

#For all the dno 12 emp's-->update the sal by 10%

```
mysql> update emp set sal=sal+sal*0.10 where dno=12;
```

```
mysql> select * from emp;
```

| eid | ename  | sal   | sex | dno |
|-----|--------|-------|-----|-----|
| 101 | Miller | 10000 | m   | 11  |
| 102 | Blake  | 29700 | m   | 12  |
| 103 | Sony   | 30000 | f   | 11  |
| 104 | Geeta  | 44000 | f   | 12  |
| 105 | James  | 50000 | m   | 13  |

#Delete:

```
mysql> delete from emp where ename="James";
```

```
mysql> select * from emp;
```

| eid | ename  | sal   | sex | dno |
|-----|--------|-------|-----|-----|
| 101 | Miller | 10000 | m   | 11  |
| 102 | Blake  | 29700 | m   | 12  |
| 103 | Sony   | 30000 | f   | 11  |
| 104 | Geeta  | 44000 | f   | 12  |

#Filtering operations:

#i)Filter those emps whose sal>25000

```
mysql> select * from emp where sal > 25000;
```

| eid | ename | sal   | sex | dno |
|-----|-------|-------|-----|-----|
| 102 | Blake | 29700 | m   | 12  |
| 103 | Sony  | 30000 | f   | 11  |

```
| 104 | Geeta | 44000 | f    | 12 |
+-----+-----+-----+-----+
```

#ii)Filter only male emp records

```
mysql> select * from emp where sex='m';
+-----+-----+-----+-----+
| eid | ename | sal   | sex  | dno  |
+-----+-----+-----+-----+
| 101 | Miller | 10000 | m    | 11  |
| 102 | Blake  | 29700 | m    | 12  |
+-----+-----+-----+-----+
```

```
#=====
```

#Groupings and aggregations:

#Various Aggregations-->sum(),avg(),max(),min(),count()

1.Single Grouping and Single Aggregation

2.Single Grouping and multiple Aggregations

3.Multi-grouping and single aggregation

4.Multi-grouping and Multiple Aggregations

#1.Single Grouping and Single Aggregation

#dnowise---->totsal

o/p: 11 --->totsal

12 --->totsal

13 --->totsal

```
mysql> select dno,sum(sal) from emp group by dno;
```

```
+-----+
| dno | sum(sal) |
+-----+
```

```
| 11 | 40000 |
| 12 | 73700 |
+-----+
```

```
#=====
```

#2.Multi-grouping and single aggregation

```
mysql> select dno,sex,sum(sal) from emp group by dno,sex;
```

```
+-----+
| dno | sex  | sum(sal) |
+-----+
```

```
| 11 | m    | 10000 |
+-----+
```

```

|   12 | m      |    29700 |
|   11 | f      |    30000 |
|   12 | f      |    44000 |
+-----+
#=====

```

### #3.single grouping and multiple aggregations

```
mysql> select dno,sum(sal),avg(sal),max(sal),min(sal),count(*) from emp group by dno;
```

```

+-----+-----+-----+-----+
| dno | sum(sal) | avg(sal) | max(sal) | min(sal) | count(*) |
+-----+-----+-----+-----+
|   11 |    40000 | 20000.0000 |    30000 |    10000 |        2 |
|   12 |    73700 | 36850.0000 |    44000 |    29700 |        2 |
+-----+-----+-----+-----+
#=====
```

### #4.Multi grouping and multiple aggregation

```
mysql> select dno,sex,sum(sal),avg(sal),max(sal),min(sal),count(*) from emp group by dno,sex;
```

```

+-----+-----+-----+-----+-----+-----+
| dno | sex  | sum(sal) | avg(sal) | max(sal) | min(sal) | count(*) |
+-----+-----+-----+-----+-----+-----+
|   11 | m    |    10000 | 10000.0000 |    10000 |    10000 |        1 |
|   12 | m    |    29700 | 29700.0000 |    29700 |    29700 |        1 |
|   11 | f    |    30000 | 30000.0000 |    30000 |    30000 |        1 |
|   12 | f    |    44000 | 44000.0000 |    44000 |    44000 |        1 |
+-----+-----+-----+-----+-----+-----+
#=====
```

### #Creating a table from another table

```
mysql> create table emp2 like emp;
```

```
Query OK, 0 rows affected (0.20 sec)
```

```
mysql> select * from emp2;
```

```
Empty set (0.01 sec)
```

here only the table structure is copied but not the data

```
mysql> describe emp2;
```

```

+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| eid   | int           | YES  |      | NULL     |      |
| ename | varchar(20) | YES  |      | NULL     |      |
| sal   | int           | YES  |      | NULL     |      |
| sex   | varchar(6)   | YES  |      | NULL     |      |
| dno   | int           | YES  |      | NULL     |      |
+-----+-----+-----+-----+
#=====
```

```
#=====
#inserting data from one table to another table
```

```
mysql> insert into emp2 select * from emp;
```

```
mysql> select * from emp2;
```

| eid | ename  | sal   | sex | dno |
|-----|--------|-------|-----|-----|
| 101 | Miller | 10000 | m   | 11  |
| 102 | Blake  | 29700 | m   | 12  |
| 103 | Sony   | 30000 | f   | 11  |
| 104 | Geeta  | 44000 | f   | 12  |

```
#=====
#Table to Table copy
#CTAS--->CREATE-TABLE-AS-SELECT
```

```
mysql> create table emp3 as select * from emp;
```

```
mysql> select * from emp3;
```

| eid | ename  | sal   | sex | dno |
|-----|--------|-------|-----|-----|
| 101 | Miller | 10000 | m   | 11  |
| 102 | Blake  | 29700 | m   | 12  |
| 103 | Sony   | 30000 | f   | 11  |
| 104 | Geeta  | 44000 | f   | 12  |

```
#=====
#Merging tables--->using union
```

```
mysql> create table emp4 as select * from emp2 union all select * from emp3;
```

```
mysql> select * from emp4;
```

| eid | ename  | sal   | sex | dno |
|-----|--------|-------|-----|-----|
| 101 | Miller | 10000 | m   | 11  |
| 102 | Blake  | 29700 | m   | 12  |
| 103 | Sony   | 30000 | f   | 11  |
| 104 | Geeta  | 44000 | f   | 12  |
| 101 | Miller | 10000 | m   | 11  |
| 102 | Blake  | 29700 | m   | 12  |

```

| 103 | Sony   | 30000 | f    | 11 |
| 104 | Geeta  | 44000 | f    | 12 |
+----+-----+-----+-----+

```

#union all--->with duplicates  
#union----->without duplicates

```
#=====
#Eliminating the duplicates using--->distinct
```

```
mysql> create table emp5 as select distinct * from emp4;
```

```
mysql> select * from emp5;
```

```

+----+-----+-----+-----+
| eid | ename | sal   | sex  | dno  |
+----+-----+-----+-----+
| 101 | Miller | 10000 | m    | 11 |
| 102 | Blake  | 29700 | m    | 12 |
| 103 | Sony   | 30000 | f    | 11 |
| 104 | Geeta  | 44000 | f    | 12 |
+----+-----+-----+-----+

```

```
#=====
#Alter:  

#i)Adding new columns----->dname and designation
```

```
mysql> alter table emp2  

-> add dname varchar(10),  

-> add designation varchar(10);
```

```
mysql> select * from emp2;
```

```

+----+-----+-----+-----+-----+-----+
| eid | ename | sal   | sex  | dno  | dname | designation |
+----+-----+-----+-----+-----+-----+
| 101 | Miller | 10000 | m    | 11  | NULL  | NULL          |
| 102 | Blake  | 29700 | m    | 12  | NULL  | NULL          |
| 103 | Sony   | 30000 | f    | 11  | NULL  | NULL          |
| 104 | Geeta  | 44000 | f    | 12  | NULL  | NULL          |
+----+-----+-----+-----+-----+-----+

```

#ii) Renaming a column using alter command

```
#      Rename eid--->ecode  

#      Rename sal--->income  

#              sex--->Gender
```

```
mysql> alter table emp2  

-> change column eid ecode int,  

-> change column sal income int,  

-> change column sex Gender varchar(10);
```

```
mysql> select * from emp2;
+-----+-----+-----+-----+
| ecode | ename | income | Gender | dno   | dname | designation |
+-----+-----+-----+-----+
| 101  | Miller | 10000 | m     | 11    | NULL  | NULL        |
| 102  | Blake  | 29700 | m     | 12    | NULL  | NULL        |
| 103  | Sony   | 30000 | f     | 11    | NULL  | NULL        |
| 104  | Geeta  | 44000 | f     | 12    | NULL  | NULL        |
+-----+-----+-----+-----+
```

#iii) dropping a particular column

```
mysql> alter table emp2
      -> drop column dname;
```

```
mysql> select * from emp2;
```

```
+-----+-----+-----+-----+
| ecode | ename | income | Gender | dno   | designation |
+-----+-----+-----+-----+
| 101  | Miller | 10000 | m     | 11    | NULL  |
| 102  | Blake  | 29700 | m     | 12    | NULL  |
| 103  | Sony   | 30000 | f     | 11    | NULL  |
| 104  | Geeta  | 44000 | f     | 12    | NULL  |
+-----+-----+-----+-----+
```

```
#=====
#iv) Rename a table
```

```
mysql> alter table emp2
      -> rename to employee;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> select * from emp2;
```

```
ERROR 1146 (42S02): Table 'mydb81.emp2' doesn't exist
```

```
mysql> select * from employee;
```

```
+-----+-----+-----+-----+
| ecode | ename | income | Gender | dno   | designation |
+-----+-----+-----+-----+
| 101  | Miller | 10000 | m     | 11    | NULL  |
| 102  | Blake  | 29700 | m     | 12    | NULL  |
| 103  | Sony   | 30000 | f     | 11    | NULL  |
| 104  | Geeta  | 44000 | f     | 12    | NULL  |
+-----+-----+-----+-----+
```

```
#=====
```

#Creating a table by copying data from other table of another database

```
mysql> create table emp6 as select * from mydb80.emp;
```

```
mysql> select * from emp6;
+----+-----+-----+-----+
| eid | ename | sal   | sex   | dno   |
+----+-----+-----+-----+
| 101 | Miller | 10000 | m    | 11   |
| 102 | Blake  | 12200 | m    | 12   |
| 103 | Sony   | 30000 | f    | 11   |
| 104 | Sita   | 14400 | f    | 12   |
+----+-----+-----+-----+
```

```
#=====
```

#Difference b/w delete Vs Truncate Vs drop

#1.delete : deletes a particular record/records  
# deletes a particular row/rows

#ex: delete all the emp recs who belongs to the dno=12

```
mysql> delete from emp3 where dno=12;
```

```
mysql> select * from emp3;
+----+-----+-----+-----+
| eid | ename | sal   | sex   | dno   |
+----+-----+-----+-----+
| 101 | Miller | 10000 | m    | 11   |
| 103 | Sony   | 30000 | f    | 11   |
+----+-----+-----+-----+
```

```
#=====
```

#2.truncate :deletes the table data but still the table structure remains

```
mysql> truncate table emp3;
Query OK, 0 rows affected (0.10 sec)
```

```
mysql> select * from emp3;
Empty set (0.01 sec)
```

```
mysql> describe emp3;
+----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+----+-----+-----+-----+-----+
| eid   | int       | YES  |     | NULL    |       |
| ename | varchar(20)| YES  |     | NULL    |       |
| sal   | int       | YES  |     | NULL    |       |
| sex   | varchar(6) | YES  |     | NULL    |       |
| dno   | int       | YES  |     | NULL    |       |
+----+-----+-----+-----+-----+
```

```

#=====
#drop :drops the entire table from the database
# i.e deletes the entire table data and also the table structure

mysql> drop table emp4;
Query OK, 0 rows affected (0.19 sec)

mysql> select * from emp4;
ERROR 1146 (42S02): Table 'mydb81.emp4' doesn't exist

#=====
#Between clause:
#I want those emps whose salaries are between 25000 and 40000
mysql> select * from emp where sal between 25000 and 40000;
+----+-----+-----+-----+
| eid | ename | sal   | sex   | dno   |
+----+-----+-----+-----+
| 102 | Blake | 29700 | m     | 12    |
| 103 | Sony   | 30000 | f     | 11    |
+----+-----+-----+-----+

#ex:2 not between
mysql> select * from emp where sal not between 25000 and 40000;
+----+-----+-----+-----+
| eid | ename | sal   | sex   | dno   |
+----+-----+-----+-----+
| 101 | Miller | 10000 | m     | 11    |
| 104 | Geeta  | 44000 | f     | 12    |
+----+-----+-----+-----+

#ex:3 between and not in
#I want those employees whose sal between 25000 and 40000 and should not
#belong to dno 11 and 13.
mysql> select * from emp where sal between 25000 and 40000 and dno not in (11,13);
+----+-----+-----+-----+
| eid | ename | sal   | sex   | dno   |
+----+-----+-----+-----+
| 102 | Blake | 29700 | m     | 12    |
+----+-----+-----+-----+

#=====
#Having clause
#Having clause with group by
#Task :I want the total revenue for each department and the revenue should be
#      more than 25000

mysql> select dno,sum(sal) from emp group by dno having sum(sal) > 25000;
+-----+

```

```
| dno | sum(sal) |
+----+-----+
|   11 |    40000 |
|   12 |    73700 |
+----+-----+
```

```
#=====
#order by:
```

```
#i)order by enames
```

```
mysql> select * from emp order by ename;
```

```
+----+-----+-----+-----+
| eid | ename  | sal   | sex   | dno  |
+----+-----+-----+-----+
| 102 | Blake  | 29700 | m    | 12   |
| 104 | Geeta  | 44000 | f    | 12   |
| 101 | Miller | 10000 | m    | 11   |
| 103 | Sony   | 30000 | f    | 11   |
+----+-----+-----+-----+
```

```
#=====
#ii)order by sal in descending order
```

```
mysql> select * from emp order by sal desc;
```

```
+----+-----+-----+-----+
| eid | ename  | sal   | sex   | dno  |
+----+-----+-----+-----+
| 104 | Geeta  | 44000 | f    | 12   |
| 103 | Sony   | 30000 | f    | 11   |
| 102 | Blake  | 29700 | m    | 12   |
| 101 | Miller | 10000 | m    | 11   |
+----+-----+-----+-----+
```

```
#=====
#limit:
```

```
mysql> select * from emp limit 2;
```

```
+----+-----+-----+-----+
| eid | ename  | sal   | sex   | dno  |
+----+-----+-----+-----+
| 101 | Miller | 10000 | m    | 11   |
| 102 | Blake  | 29700 | m    | 12   |
+----+-----+-----+-----+
```

```
#=====
#A Query consisting of
```

```
1.group by
```

```
2.having
```

```
3.order by
```

#### 4.limit

```
#ex: I want top 2 departments with highest revenue(consider only those
#      depts whose revenue should be bove 30000)
#1.group by
#2.having
#3.order by
#4.limit
```

```
mysql> select dno,sum(sal) as revenue from emp group by dno having revenue>30000 order by revenue
desc limit 2;
+----+-----+
| dno | revenue |
+----+-----+
|   12 |    73700 |
|   11 |    40000 |
+----+-----+
```

#Python Database Connectivity:

#Python to connect with any database,we require a module

#Python to connect with mysql---->we require pymysql module

#Python to connect with oracle--->we require cx\_Oracle module

#Python to connect with sqlserver->we require pyodbc module

#Downloading and Installing pymysql module

#Pymysql is an external module

#External module :A Module which is not part of python software,such module
# we call it as external module

#These external modules will be installed by using a component called pip

#pip :preferred installer program

#pip is a package manger which quickly downloads and installs any

#external module

#pip is available within the installation folder of python

#i.e C:\python313\scripts\pip

#To check where python avialble in ur system-->Goto cmd prompt-->say

C:\Users\DELL>where python

```
C:\Python314\python.exe  
#C:\Users\DELL>cd\  
C:\>cd python313\scripts  
C:\Python313\Scripts>pip install pymysql  
=====  
#python to connect with any database and to perform sql operations,  
#we require 2 objects  
  
#1.connection object:  
-If we call connect() function, by providing the following 4 connection parameters  
then connection object is created  
1.host name  
2.username of mysql  
3.password of mysql  
4.databasename  
  
#ex:  
import pymysql  
con=pymysql.connect(host="localhost",user="root",password="root",database="mydb81")  
Here con is the connection object  
#2.cursor object:  
-on the connection object, if we pass cursor method, then cursor object  
is created  
  
cur1=con.cursor() #here cur1 is the cursor object.  
  
#Why cursor object  
  
within the cursor object(cur1)-->we have execute() method,  
within the execute method-->we provide any valid SQL query  
ex:  
cur1.execute("select * from emp")  
  
The o/p rows of the sql query is stored within the cur1 object  
  
Apply for loop and retrieve one by one row  
  
for row in cur1:  
    print(row)
```

```
#Create a database(mydb82) within the mysql

import pymysql

#Creating connection object
con=pymysql.connect(host="localhost",user="root",password="root")
#here con is the connection object

#Creating cursor object
cur1=con.cursor() #here cur1 is the cursor object
cur1.execute("create database mydb82;")
print("DATABASE CREATED SUCCESSFULLY!!!")

cur1.close()

con.close()
```

```
#creating multiple cursor objects
#cur1 ---->For creating table
#cur2 ---->for inserting data
#cur3 ---->for retrieving data

import pymysql

#Creating connection object
con=pymysql.connect(host="localhost",user="root",password="root",database="mydb82")

#Creating cur1 object for creating table
cur1=con.cursor()
cur1.execute("Create table if not exists emp(eid int,ename varchar(20),sal int,sex varchar(10),dno int)")
print("TABLE CREATED SUCCESSFULLY!!!")
#Creating cur2 object for inserting data into table
cur2=con.cursor()
cur2.execute("insert into emp
values(101,'Miller',10000,'m',11),(102,'Blake',20000,'m',12),(103,'Sony',30000,'f',11),(104,'Geeta',40000,
'f',12),(105,'James',50000,'m',13)")
print("DATA INSERTED SUCCESSFULLY!!!")
con.commit() #To save permanently

#Creating cur3 object for retrieving data
cur3=con.cursor()
cur3.execute("select * from emp")
```

```
#The o/p rows of the sql stmt is stored within cur1 object  
#Now retrieving one by one row using for loop
```

```
for row in cur3:  
    print(row)
```

```
cur1.close()  
cur2.close()  
cur3.close()
```

```
con.close()
```

```
#In the above example instead of 3 cursor objects,  
#we can also perform the same using single cursor object
```

```
#creating multiple cursor objects  
#cur1 ---->For creating table  
#cur1 ---->for inserting data  
#cur1 ---->for retrieving data
```

```
import pymysql
```

```
#Creating connection object  
con=pymysql.connect(host="localhost",user="root",password="root",database="mydb82")
```

```
#Creating cur1 object for creating table  
cur1=con.cursor()  
cur1.execute("Create table if not exists employee(eid int,ename varchar(20),sal int,sex varchar(10),dno  
int)")  
print("TABLE CREATED SUCCESSFULLY!!!!")
```

```
cur1.execute("insert into employee  
values(101,'Miller',10000,'m',11),(102,'Blake',20000,'m',12),(103,'Sony',30000,'f',11),(104,'Geeta',40000,  
'f',12),(105,'James',50000,'m',13)")  
print("DATA INSERTED SUCCESSFULLY!!!!")  
con.commit() #To save permanently
```

```
cur1.execute("select * from employee")  
#The o/p rows of the sql stmt is stored within cur1 object  
#Now retrieving one by one row using for loop
```

```
for row in cur1:  
    print(row)
```

```
cur1.close()

con.close()

#when to go for multiple cursor objects

#ex:
cur1.execute("select * from employee")

cur1.execute("select * from customer")
```

What we see in cur1 object???----> customer data

but in your program code, you want both employee and customer data to be retrieved at any time , go for multiple curosrs objects as

```
cur1.execute("select * from employee") --->here cur1-->stores employee data

cur2.execute("select * from customer")--->here cur2-->stores customer data
```

```
#Update operation

#Python code to update all the dno 11 emps salaries with a hike of 20%

import pymysql

#Creating connection object
con=pymysql.connect(host="localhost",user="root",password="root",database="mydb82")

#Creating cur1 object for creating table
cur1=con.cursor()
cur1.execute("update emp set sal=sal+sal*0.20 where dno=11")
con.commit()
print("DATA UPDATED SUCCESSFULLY!!!!")

cur1.execute("select * from emp")
for row in cur1:
    print(row)

cur1.close()
con.close()
```

```
#delete operation

#Python code to delete a record

#Delete operation
import pymysql

#Creating connection object
con=pymysql.connect(host="localhost",user="root",password="root",database="mydb82")

#Creating cur1 object for creating table
cur1=con.cursor()
cur1.execute("delete from emp where eid=105")
con.commit()
print("DATA DELETED SUCCESSFULLY!!!")

cur1.execute("select * from emp")
for row in cur1:
    print(row)

cur1.close()
con.close()
```

```
#Python code to create a table from another table
import pymysql

#Creating connection object
con=pymysql.connect(host="localhost",user="root",password="root",database="mydb82")

#Creating cur1 object for creating table
cur1=con.cursor()
cur1.execute("create table emp2 like emp")

#inserting data from one table to another
cur1.execute("insert into emp2 select * from emp")
con.commit()

#Retrieving data
cur1.execute("select * from emp2")
```

```
for row in cur1:  
    print(row)  
  
cur1.close()  
  
con.close()  
  
  
#Table to Table copy  
#CTAS :CREATE TABLE as Select  
  
#Python code to create a table from another table  
import pymysql  
  
#Creating connection object  
con=pymysql.connect(host="localhost",user="root",password="root",database="mydb82")  
  
#Creating cur1 object for creating table  
cur1=con.cursor()  
cur1.execute("create table emp3 as select * from emp")  
  
#Retrieving data  
cur1.execute("select * from emp3")  
  
for row in cur1:  
    print(row)  
  
cur1.close()  
  
con.close()  
  
  
#merging tables--->using union all  
#Python code to create a table from another table  
import pymysql  
  
#Creating connection object
```

```
con=pymysql.connect(host="localhost",user="root",password="root",database="mydb82")

#Creating cur1 object for creating table
cur1=con.cursor()
cur1.execute("create table empres as select * from emp2 union all select * from emp3")

#Retrieving data
cur1.execute("select * from empres")

for row in cur1:
    print(row)

cur1.close()

con.close()
```

```
#multiple methods for fetching data:3 methods
#fetchone()
#fetchmany(n)
#fetchall()

import pymysql

#Creating connection object
con=pymysql.connect(host="localhost",user="root",password="root",database="mydb82")

#Creating cur1 object for creating table
cur1=con.cursor()

#Retrieving data
cur1.execute("select * from emp3")

=====
#1.fetchone(): To fetch one record at a time

row1=cur1.fetchone() #It fetches the 1st record
print(row1)

row2=cur1.fetchone() #It fetches the 2nd record
```

```

print(row2)
print("\n")
#=====
#2.fetchmany(n):It fetches 'n' records
#cur1.execute("select * from emp3")

rows=cur1.fetchmany(3) #here rows is a tuple of 3 recs
print(rows)
print("\n")
#=====
#3.fetchall():It fetches all the records
cur1.execute("select * from emp3")
rows=cur1.fetchall()
print(rows)
#=====
cur1.close()

con.close()

```

```

import pymysql

#Creating connection object
con=pymysql.connect(host="localhost",user="root",password="root",database="mydb82")

#Creating cur1 object for creating table
cur1=con.cursor()

#Retrieving data
cur1.execute("select * from emp")

#Here we are getting the output in the form of tuples,
#built in the form of list
for row in cur1:
    print(list(row))
#=====


```

```
#If you want to modify the data after retrieving
#store all the records as part of a list, so that u can modify
cur1.execute("select * from emp")
y=[]
for row in cur1:
    y.append(list(row))
print(y)
y[1][2]=25000
print(y)
cur1.close()

con.close()
```

