**Core Python**

Session 1

Python is used for:

1)Automation

2)Data Analysis (Anlysis and Visualization)

3)Scientific

4)Web Applications

5)Test Cases

6)Networking and IOT(Internet of Tracing) Applications

7)Admin Activities

8)GUI Applications

9)Gaming Applications

10)Animation Applications

a)Procedure Oriented Languages Ex:- C

b)OOPS(Object Oriented Programs) Ex:- C++, Java

c)Scripting Languages

Ex:- shell scripting,Python scripting,javascript

d)Modular Programing Languages Ex:- Modular-2 and Modular-3

89,300 modules are available in Python

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

Session 2

Python is a general purpose Language.

Python is developed by the G.V.R in 1989 at CWI.

CWI is a mathematics and science Reasearch center located at Netherlands.

G.V.R implemented Python Language by taking different validaties of Languages and features.

1)Procedure Oriented Programing Lnguage(C)

2)Object Oriented Programing Lnguage(CPP)

3)Scripting Programing Lnguages(shell scrip and java script)

4)Modular Programing Lnguages(Modules,Modular3)

Object Oriented Programs contains

a)Encapsulation

b)Polymorphism

c)Inheritance

d)Abstraction

If we use above concepts in the programing then we get

i)security

ii)flexibility and

iii)reusability

Scripting Languages are meant for write and run(Javascript,Python and PHP)

Programing Languages are meant for write, compile and run(C,Java,.Net)

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

Session 3

Python is used to implement different verities of applications like:

1)Automation Applications

2)Data Analytics

3)Scietific Applications

4)Web Applications

5)Test Cases

6)Networking and IOT(Internet of Tracing) Applications

7)Administration Script

8)GUI Applications

9)Gaming Applications

10)Animation Applications and so on

**Features of python**

\* Python is simple and easy to learn.

\* Python is Opensource.

\*Python is high level Language.

\*Python is platform Independent(Works on Windows/Linux/MAC/Solaris etc) Python3.5 is developed seperately for every OS. Python is portable(We can run on Any Operating system)

\*Python is embeded.

\* Python is extensible(Other Language code we can use in Python)

\* Python is redistributed

\* Pyhon is implemented using OOPS(Object Oriented Programing)

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

Session 4

Author of Python is PSF

PSF-->Python Software Foundation

CPython

JPython(D1) Anaconda(D2) IPython(D3)

CPython is Original Python

D1,D2,D3 are meant for Distribution

**Python Contains**

\* Dynamic Datatypes

\* Interpretation(Need not compile explicitly)

\* Exception Handling

\* Multi-Threading

\* Modules(Pre-defined Programs)

Python 3.6(Released on 23rd December 2016)

Python 2.7 is most Popular

Python 3.5(June 2016) is most stable version

By default Python is available in Linux OS(Operating System)

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

Session 5

**Instalation of Python**

Download the latest version of Python Software(Python3.6) from the [www.python.org](http://www.python.org/) website.

With respect to every OS seperate Python softwares are available.

For 32-bit and 64-bit OS seperate Python softwares are available.

After downloading the **python-3.6.0.exe** double click on **python-3.6.0.exe** file

Then click on **run**

Then click on customize **installation**

Then click on **NEXT**

Select the location in which Drive you want to install the python software

Then click on **Install**

Then click on **Close**

By default Python software contains the Python interpretor.

By default we can access the Python interpretor within the installation folder of the Python software only.

Inorder to access the Python interpretor from any directory of the OS we need to set the path.

Path is a environment variable which is related to OS.

D:\>python -v

'python' is not recognized as an internal or external command,

operable program or batch file.

step1: Right click on **MyComputer**

step2: click on **Properties**

step3:click on **Advanced System Settings**

step4: click on **Advanced** tab

step5: click on **Environment variables**

step6: Go to System Variables

select the **path**

Click on **Edit**

Add the following Path at the end of the variable value.

**C:\Python27**

Then click on **OK**

Then click on **OK**

Then click on **OK**

D:\>python -v

>>>3+4

7

>>>"Welcome to Python"

'Welcome to Python'

>>>"HI"\*3

'HI HI HI'

>>>x=20

>>>y=30

>>>x+y

50

>>>quit()

D:\>python

>>>x+y

Error

Python are files are saved with .py extension

sample.py

x=25

y=20

print(x+y)

print(x-y)

print(x\*y)

print(x/y)

print(x%y)

D:\>python sample.py

45

5

500

1

5

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

Session 6

Python applications can be developed in two modes.

1)Interactive Mode

2)Batch Mode

**1)Interactive Mode:-**

Submitting one by one Python statement to the Python interpretor by the programer explicitly is known as a interactive mode. Inorder to develop the Python programs in interactive mode we have to open the Python interpretor. We can open the Python interpretor by running Python command on the command prompt.

D:\>python

>>>2+4

6

>>>"Hello World!"

'Hello Wolrld!'

>>>x=10

>>>y=20

>>>x+y

30

>>>quit()

D:\>

Interactive mode is suitable for learning the python but it is not suitable for developing the projects or applications because whenever we exit from python interpretor the work which is done previously is going to be loss.

**2)Batch Mode:-**

Writing the group of statements in afile, saving that file with the extension of .py and submiting that file to the python interpretor at a time is known as a batch mode.

Inorder to develop the python programs in batch mode we use Editors or IDE's. The various Editors are:

a)Notepad

b)Notepad++

c)EditPlus

d)gedit

e)vi

f)vim

g)kedit

h)kwrite

i)nano etc

Different IDE's are:

a)Pycharm

b)Eric

c)wing

d)NetBeans

e)Eclipse

test.py

x=100

y=20

print(x+y)

print(x-y)

print(x\*y)

print(x/y)

D:\>python test.py

120

80

2000

5

If we develop the Python applications by using Editors then development time,cost will be increased, automatic debugging is not possible and we cannot able to use the version controlling tools.

To overcome the above problems we use the IDE's for the development of Python applications or Projects.

Session 7

**demo.py**

x=input("Enter first number:")

y=input("Enter second number:")

print(x+y)

$nano demo.py

ctrl+x to exit from nano editor

**nano** editor is available in Linux environment

IDLE(Integrated Development Learning Environment) is Python IDE.

x=1000

y=200

print(x+y,x-y,x\*y,x/y)

Any Organisation usecase or scenario contains two parts

1)Data

2)Operations

|------->cid,cname,caddr,cval

|-------Customer-->|

| |--------->deposit(),withdraw(),balenq()

Bank--------> |

| |-------->eid,ename,esal,eaddr

|-------Employee-->|

|-------->hra(),ta(),da(),pf(),tax(),tsal()

Data of the any Oraganization usecase can be represented in any programing Language by using data types and variables. Every programming Language supports the datatypes and variables. But datatypes and variables of one programming Language are not going to be same with the datatypes and variables of another programming Language.

Every programming Language provides functions or methods or both to represent the operations.

**Datatypes**

Programming Languages can support either static datatypes or dynamic datatypes.

**static datatypes**

In the static datatypes programer should define datatype of the variable explicitly and store the data into the variable. Once the datatype is defined by the programer for a variable then in the entire program we cannot change the datatype of that variable. C,CPP,Java,>Net Langauges supports static datatypes.

**dynamic datatypes**

In the dynamic datatypes we need the execution of the program. Datatype of variable will be decided based on the data which is assigned to the variable. During the execution of the program datatype of the variable can be modified. Python,javascript and PHP are supporting dynamic datatypes.

Session 8

**Python datatypes:-**

Every datatype in Python Language represents a class. Whenever we assigned data to the variable at the time of execution of the program corresponding data represented datatype class Object will be created and that Object address will be assigned to the variable.

Two different Objects doesnot contain the same address(hashcode value).

Python datatypes are categorised as:

1)Fundamental datatypes

2)Collection datatypes

**1)Fundamental datatypes**

Fundamental datatypes represented classes Objects can store only one value at a time. Python supports the following fundamental types

a)int

b)float

c)complex

d)bool

e)str

**2)Collection datatypes**

Collection datatypes represented classes Objects can store more than one value at a time. Python supports the following Collection types

a)list

b)set

c)tuple

d)dict

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

x=1000

print(type(x))

print(id(x))

print(x)

x=2000

print(type(x))

print(id(x))

print(x)

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

a=1000

b=34.56

c=3+4j

d=True

e='python'

print(type(a))

print(a)

print(type(b))

print(b)

print(type(c))

print(c)

print(type(d))

print(d)

print(type(e))

print(e)

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

Session 9

**Types of Objects:-**

Python supports two types of Objects. They are:

1)Immutable Objects

2)Mutable Objects

**1)Immutable Objects**

The Objects which does not allow to modify the content are knownas Immutable Objects. Whenever we try to modify the data of the Immutable Objects without modifying the data of those Objects new Objects will be created with the modified content.

Python immutable Objects are:

a)int

b)float

c)complex

d)bool

e)str

f)tuple

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

x=10

print(x)

print(id(x))

x=10

print(x)

print(id(x))

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

Immutable Objects will give better performance when compared to mutable Objects. We cannot create two different immutable Objects with the same content at any given point of time

x=10

print(x)

print(id(x))

y=10

print(y)

print(id(y))

Applying the iterations on immutable Objects will give better performance.

**2)Mutable Objects**

The Objects which we allows to modify the content are known as Mutable Objects.

Python mutable Objects are:

a)list

b)set

c)dict

x=[10,20,30]

print(x)

print(id(x))

x[1]=12

print(x)

print(id(x))

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

We can create two different mutable Objects with the same content

x=[10,20,30]

print(x)

print(id(x))

y=[10,20,30]

print(y)

print(id(y))

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

Session 10

**Number System**

a)Bynary Number System

b)Octal Number System

c)Decimal Number System

d)Hexa-Decimal Number System

**a)Binary Number System**

The base or radix of binary number system is 2.

The possible digits that are used in binary number system are 0 and 1

We can give the input values in the form of binary numbers format by giving ob as prefix.

x=0b1010

1010=1\*2\*2\*2 + 0\*2\*2 + 1\*2 + 0\*1

=8+0+2+0

=10

y=0b1020 #error

x=ob1010

print(x)

**b)Octal Number System**

The base or radix of octal number system is 8.

The possible digits that are used in octal number system are 0 to 7.

We can give the octal number system number as input by giving 0(zero) begining of the number.

x=0123

123 =1\*8\*8 + 2\*8 + 3\*1

=64+16+3

=83

y=0128 #error

z=0123

print(z)

**c)Decimal Number System**

The base or radix of decimal number system is 10.

The possible digits that are used in decimal number system are 0 to 9.

The default number system followed by the python is decimal number system.

x=1234

1234=1\*10\*10\*10 + 2\*10\*10 + 3\*10 + 4\*1

=1000+ 200 + 30 + 4

=1234

x=12c4 #error

x=1234

print(x)

**d)Hexa-Decimal Number System**

The base or radix of hexa-decimal number system is 16.

The possible digits that are used in hexa-decimal number system are 0 to 9 and A to F

We can give the hexa-decimal number system number as input by giving 0X as prefix.

x=0x25

25=2\*16 + 5\*1

=32+5

=37

x=0X2c=2\*16 + 12\*1

=32+12

=44

x=0x3Z #error

x=0X25

print(x)

We can give any number system number as input but we will get the output in the form of decimal number system format by default.

x=0b1010

print(x)

print(type(x))

y=0123

print(y)

z=345

print(z)

w=0x25

print(w)

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

By using pre-defined functions we can display the any number in the form of any number system format.

x=0b1010

print(x)

print(bin(x))

print(oct(x))

print(hex(x))

x=0123

print(x)

print(bin(x))

print(oct(x))

print(hex(x))

x=1234

print(x)

print(bin(x))

print(oct(x))

print(hex(x))

x=0X25

print(x)

print(bin(x))

print(oct(x))

print(hex(x))

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

**str**

Group of characters or sequence of characters are known as String.

We can represent the Strings in Python by using single quotes or by using double quotes or triple quotes.

Single quotes are used to represent one line String, triple quotes are used to represent multiple lines. Every character in the String Object is represented with unique index.

Python Strings are supporting both the +ve indexing and -ve indexing.

+ve index starts with 0(zero) whereas -ve index starts from -1

x='python'

print(x)

y='''Java

is a programing

Language'''

print(y)

z="""python is a

procedure oriented as well as

object oriented Language"""

print(z)

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

Session 11

x='python'

print(x)

print(x[2])

print(x[-3])

print(x[2:5])

print(x[2:])

print(x[5:1])

print(x[-5:-2])

print(x[-3:-5])

print(len(x))

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

x='python'

y='lang'

z=x+y

print(x)

print(id(x))

print(y)

print(id(y))

print(x)

print(y)

print(z)

x=x+y

print(id(x))

print(x)

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

x='python'

print(x)

print(id(x))

y='lang'

print(y)

print(id(y))

x=x+y

print(x)

print(id(x))

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

**Reading data from the Keyboard:-**

We can read the data from the keyboard by using input function.

x=raw\_input('enter fname:')

print(x)

y=raw\_input('enter lname:')

print(y)

If we use Python 2.7 version for taking input as String datatype then we must use raw\_input whereas for other datatypes we can use as below.

a=input("Enter int value:")

i=int(a)

print(type(i))

print(i)

b=input("Enter float value:")

j=float(b)

print(type(j))

print(j)

c=input("Enter complex value:")

k=complex(c)

print(type(k))

print(k)

d=input("Enter boolean value:")

l=bool(d)

print(type(l))

print(l)

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

Input function by default takes the any type of data in the form of String format only. After reading the data in the form of String format later we can convert the data in the form of required type conversion functions.

x=raw\_input('enter fno:')

print(type(x))

i=int(x)

print(type(i))

y=raw\_input('enter sno:')

print(type(y))

j=int(y)

print(type(j))

print('x+y=',x+y)

print('i+j=',i+j)

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

Session 12

**Operators**

Operators are the constructs which are used to perform the operations on the data of the operators.

Python supports below operators

a)Arithmetic

b)Comparisioin

c)Logical

d)Bitwise

e)Assignment

f)Special Operators

**a)Arithmetic**

These are used to perform mathematical operations by addition, subtraction and so on. Different arithmetic operators which are supported by the Python are:

1)Addition(+)

2)Subtraction(-)

3)Multiplication(\*)

4)Division(/)

5)Modulus(%)

6)floor division(//)

7)exponent(\*\*)

x=10

y=3

print(x+y)

print(x-y)

print(x\*y)

print(x/y)

print(x%y)

print(x//y)

print(x\*\*y)

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

**b)Comparisioin and Relational Operators**

Comparision Operators are used to compare the data of the operands. After comparing the data comparision operators returns either True or False values.

Differenet comparision operators which are supported by the Python are:

a)>

b)<

c)>=

d)<=

e)==

f)!=

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

x=10

y=21

print('x>y is:',x>y)

print('x>=y is:',x>=y)

print('x<y is:',x<y)

print('x<=y is:',x<=y)

print('x==y is:',x==y)

print('x!=y is:',x!=y)

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

a='python'

b='python'

print(a==b)

print(a is b)

p=[11,22,33]

q=[11,22,33]

r=[10,20,30]

print(p==q)

print(p==r)

print(p is q)

print(p is r)

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

x='python'

print(x)

print(id(x))

y='python'

print(y)

print(id(y))

print(x==y)

print(x is y)

print(x is not y)

print("------------")

p=[11,22,33]

print(p)

print(id(p))

q=[11,22,33]

print(q)

print(id(q))

print(p==q)

print(p is q)

print(p is not q)

print("------------")

i=True

j=False

print(i is j)

print(i is not j)

print(i or j)

print(not i)

print(not j)

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

**c)Logical**

These are used to perform the mathematical and logical operations. Logical operators can be applied on boolean type operands only.

a)and

b)or

c)not

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

x=True

y=False

print(x and y)

print(x or y)

print(not x)

print(not y)

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

**d)Bitwise**

These operators will operate on consecutive bits.

a)&

b)|

c)^

a=9----->1001

b=10--->1010 &

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

1000

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

a=9----->1001

b=10--->1010 |

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

1011

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

a=9----->1001

b=10--->1010 ^

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

0011

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

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

a=9

b=10

print(a&b)

print(a|b)

print(a^b)

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

Session 13

**e)Assignment**

Python will not support post increment and post decrement operators.

In place of post increment and post decrement we can use assignment operators.

a)=

b)+=

c)-=

d)\*=

e)/=

f)%=

g)//=

h)\*\*=

i)&=

j)|=

k)^=

l)>>=

m)<<=

**f)Special Operators**

There are two types of special Opearors

1)Identity operators

2)Membership operators

Python supports two verities of special operators as shown above

**1)Identity operators**

These are used to compare the addresses(references) of the Object. Python supports two identity operators

a)is

b) is not

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

x='python'

y='python'

print(a is y)

orint(id(x))

print(id(y))

print(x)

print(y)

print(x is not y)

p=[10,20,30]

print(p)

print(id(p))

q=[10,20,30]

print(q)

print(id(q))

print(p is q)

print(p is not q)

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

**2)Membership operators**

These are used to search for the required element in the given sequence object.

Python supports two membership operators

i)in

ii)not in

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

x='python'

print(x)

print('t' in x)

print('t' not in x)

print('z' in x)

print('z' not in x)

y=[10,20,30]

print(y)

print(20 in y)

print(20 not in y)

print(50 in y)

print(50 not in y)

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

**Precedence of Operators**

While eveluating the expressions we have to follow the operators precedence mechanism to get the correct results.

a=20

b=10

c=15

d=5

e=0

e=(a+b)\*c/d

print("value of (a+b)\*c/d is: ",e)

e=((a+b)\*c)/d

print("value of ((a+b)\*c)/d is: ",e)

e=(a+b)\*(c/d)

print("value of (a+b)\*(c/d) is: ",e)

e=a+(b\*c)/d

print("value of (a+b)\*c/d is: ",e)

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

Session 14

**Control Flow Statements**

Control flow statements are used to disturb the normal flow of the execution of the program.

Control flow statements are categorised as:

1)Conditional Statements

2)Looping Statements

**1)Conditional Statements**

Conditional statements are used to design whether block hs to execute or skip.

**Syntax**

if <condition>:

statement1

statement2

statement3

statement4

statement5

statement6

The set of statements which are following same space indentation is known as a block/clause.

a)Block begins with when identation increases

b)Block can contain another block

c)Block ends with when the identation decreases to zero or to its containing block identation.

statement1

statement2

statement3

statement4

statement5

statement6

statement7

statement8

statement9

After evaluating any expression if we are getting the result as a boolean value then we call that expression as a condition.

There are three types of conditional statements supported by Python. They are:

a)if

b)else

c)elif

**a)if**

i) if <condition>: statement

ii)if <condition>:

statement1

statement2

If the condition returns True value then only if block will be executed otherwise it skips the execution of the if block.

print("begin")

x=int(input("Enter a +ve number:")

if x<10:

print("It is one digit number")

print("end")

**b)else**

else block execution depends on the else block preceding block condition

If the else block preceding block condition returns false value then only else block will be executed otherwise it skips the execution of that block.

print("begin")

x=int(input("Enter a +ve number:")

if x<10:

print("It is one digit number")

else:

print("Entered number is >9")

print("end")

**c)elif**

Control goes to the else-if block if it's preceding **if** block condition returns False value. After reaching the control to the elif, if **elif** block condition returns True value then only **elif** block will be executed.

print("begin")

x=int(raw\_input("enter positive number: "))

if x<10:

print("enterd number is single digit")

elif x<100:

print("enterd number is double digit")

elif x<1000:

print("enterd number is triple digit")

else:

print("enterd number greater than 1000")

print("end")

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

Session 15

**2)Looping Statements**

These are used to execute the set of statements repeatedly. Python supports following types of Looping statements.

1)while

2)for

**1)while**

It executes the set of statements repeatedly until the condition becomes false.

Syntax:

while <condition>:

statement1

statement2

statement3

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

x=1

while x<=10:

print("Welcome:",x)

x+=1

print("---------------------")

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

n=1

sum=0

while n<=10:

sum=sum+n

n+=1

print(sum)

print("---------------------")

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

a=0

while True:

print("Welcome:",a)

a+=1

if a==5:

break

print("---------------------")

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

**break:**

Inorder to exit from the infinite loop use break statement.

print("begin")

x=1

while x<=5:

print("Welcome:",x)

x+=1

else:

print("in while else block")

print("end")

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

print("begin")

x=1

while True:

print("Welcome:",x)

if(x==5):

break

x+=1

else:

print("In while block of while")

print("end")

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

print("begin")

x=0

while x<=10:

x+=1

if x==5:

continue

print("Welcome:",x)

print("end")

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

**Continue:**

It is used to skip the execution of the some part of the particular iteration.

while True:

name=raw\_input("Enter name:")

print(name)

print(type(name))

if name!='veer':

continue;

pass1=raw\_input("Enter Password:")

if pass1=='rao':

break

print("Access Granted")

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

**2)for**

for loop executes the some statements or block with respect to every element of the given sequence or iterable object.

**syntax:**

for <var\_name> in <sequence>/<iterable object>

**Ex:-**

x='python'

for p in x:

print(p)

for p in x:

print(p\*3)

**range(<num>)**

It is a pre-defined function, which is used to generate the group of values based on the given range.

n=range(10)

sum=0

for i in n:

sum+=i

print(sum)

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

Session 16

a=range(10)

for p in a:

print(p)

a=range(10,20)

for p in a:

print(p)

a=range(10,30,2)

for p in a:

print(p)

a=range(10,0,-2)

for p in a:

print(p)

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

str,list,tuple,set and dict are the classes.

len() method can be called on any of the above classes.

help() is a pre-defined function which is used to know the information about given class.

Ex:-

help(str)

help(str.title)

**EX:-**

x='python tech'

print(x)

print(x.capitalize())

print(x.upper())

print(x.lower())

print(x.title())

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

x="abc 123 xyz 456 pqr 789"

print(x)

print(x.split())

x="abc@123@xyz@456@pqr@789"

print(x)

print(x.split('@'))

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

x="Python is Opensource Language"

print(x)

print(x.split())

x="Python-is-Opensource-Language"

print(x)

print(x.split('a'))

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

Session 17

**Collections**

list,set,tuple and dict comes under Collections datatypes.

Collections represents group of elements into a single entity. Python supports the following types of Collections

1)list

2)tuple

3)set

4)dict

Every Collection type provides some methods to perform the operations on the elements of Collection Objects

**1)list**

list object can be created by using [] or by calling list function.

list object is a mutable object.

The elements of the list can be mutable or immutable.

Every element in the list is represented with unique index.

list is supporting both +ve and -ve indexing.

Insertion order is preserved.

Duplicate elements are allowed.

Heterogeneous elements are allowed.

a=[]

print(a)

print(type(a))

print(len(a))

b=list()

print(b)

print(type(b))

print(len(b))

c=[11,22,33,44]

print(c)

print(type(c))

print(len(c))

d=[100,12.34,True,'python',2+3j]

print(d)

print(type(d))

print(len(d))

e=[100,200,100,300,200,100]

print(e)

print(type(e))

print(len(e))

print(e[3])

print(e[-2])

print(e[2:5])

print(e[2:])

print(e[-3:-1])

e[2]=123

print(e)

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

x=list('sathya')

print(x)

print(len(x))

x[0]='p'

x[1]='y'

x[4]='o'

x[5]='n'

print(x)

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

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

print(x)

i=int(raw\_input("Enter Element: "))

if i in x:

print("Found")

else:

print("Not Found")

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

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

print(x)

sum1=0

for p in x:

sum1=sum1+p

print(sum1)

i=0

sum2=0

while i<len(x):

sum2+=x[i]

i+=1

print(sum2)

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

l=[11,12.23,True,'hyd',4+6j]

print(l)

print(type(l))

a,b,c,d,e=l

print(a,type(a))

print(a,type(b))

print(a,type(c))

print(a,type(d))

print(a,type(e))

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

x=[[10,20,30],[40,50,60],[70,80,90]]

print(x)

print(type(x))

print(id(x))

for i in x:

print(type(i))

for j in i:

print(j)

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

Session 18

**Working with Methods**

x=[10,40,20,50]

print(x)

print(id(x))

x.append(30)

print(x)

print(id(x))

x.insert(2,60)

print(x)

y=[11,22,33]

x.extend(y)

print(x)

print(x.index(20))

x.remove(22)

print(x)

x.sort()

print(x)

x.reverse()

print(x)

print(x.pop())

print(x)

x.pop(3)

print(x)

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

**list comprehension:**

Generating the elements into the list object by using for loop is known as a list comprehension.

x=[p for p in range(10)]

print(x)

**(OR)**

x=[]

for p in range(10):

x.append(p)

print(x)

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

n=[p for p in range(10) if p%2==0]

print(n)

**(OR)**

m=[]

for p in range(10):

if p%2!=0:

m.append(p)

print(m)

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

v=[2\*\*i for i in range(11)]

print(v)

s=[i\*\*2 for i in range(11)]

print(s)

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

noprimes=[]

count=0

for i in range(2,8):

for j in range(i\*2,50,i):

print("i=",i," j=",j)

noprimes.append(j)

count+=1

print("count=",count)

print(noprimes)

primes=[]

for x in range(2,50):

if x not in noprimes:

primes.append(x)

print(primes)

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

noprimes=[j for i in range(2,8) for j in range(i\*2,50,i)]

print(noprimes)

primes=[x for x in range(2,50) if x not in noprimes]

print(primes)

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

Session 19

**2)tuple**

It can be created by using parenthesis i.e (). OR by using tuple function.

tuple is a immutable object.

tuple supports +ve and -ve indexes.

Insertion order is preserved.

Duplicate elements are allowed.

Heterogeneous elements are allowed.

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

x=tuple()

print(x)

print(type(x))

print(len(x))

y=(10,40,20,30,10,20,1.1,True,3+4j)

print(y)

print(type(y))

print(len(y))

z=11,22,33

print(z)

print(type(z))

print(len(z))

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

x=(10,20,30,40,50)

print(x)

print(x[2])

print(x[-2])

print(x[2:5])

print(x[5:2])

y=tuple('python')

print(y)

p=(100,200,300)

print(p)

a,b,c=p

print(a)

print(b)

print(c)

print(200 in p)

print(400 in p)

print(200 not in p)

print(400 not in p)

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

x=(10,20,30,40,50)

sum=0

for i in x:

sum=sum+i

print(sum)

sum1=0

i=0

while i<len(x):

sum1=sum1+x[i]

i=i+1

print(sum1)

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

x=((11,22,33),(44,55,66),(77,88,99))

for i in x:

print(type(i))

for j in i:

print(j)

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

tuple is immutable object.

But the elements of the tuple can be mutable or immutable.

x=((10,20,30),[40,50,60],(70,80,90))

print(x)

for p in x:

print(p,type(p))

for q in p:

print(q)

x[1][2]=555

print(x)

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

x=(10,20,30,40,10,60,10,20)

print(x.count(10))

print(x.index(10))

print(x.index(10,5))

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

x=(p for p in range(10))

for q in x:

print(q)

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

Session 20

Comprehension is not supported in immutable objects.

If we try to apply the tupple comprehension we will not get the tuple object. But we will get the generator object.

x=(p for p in range(10))

print(x)

for q in x:

print(q)

**Differences between list and tuple:**

**list tuple**

1)These are mutable objects These are immutable objects

2)Applying the iterations on Applying the iterations on the

the list objects takes longer tuple objects takes less time.

time

3)list cannot be used as a key tuple can be used as a key for

for the dictionary eventhough the dictionary if the tuple

list contains immutable elements contains mutable elements.

4)list objects are not write tuple objects are write protected. protected.

5)If the frequent operations are If the frequent operations are insertion,updation and deletion retrieval of the elements then it then is recomended to go for list. is recomended to use tuple.

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

**3)set**

Set object can be created by using curly brackets i.e {}

OR by calling set function.

Insertion order inset is not preserved.

Duplicate elements are not allowed.

Heterogeneous elements are allowed.

Set is a mutable object but the elements must be immutable.

Set object doesnot support indexing.

We can perform the mathematical set operations like union, intersection, difference, symmetric difference on set objects.

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

x={10,20,30,40,50,10,20,10,12.34,True,'Python'}

print(x)

print(type(x))

y=set()

print(y)

print(type(y))

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

a=set('python')

print(a)

print(len(a))

b={10,20,30,(11,22,33)}

print(b)

for p in b:

print(p,type(p))

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

list inside list is allowed

tuple inside tuple is allowed

But set inside set is not allowed

x={10,20,30,40}

sum=0

for p in x:

sum=sum+p

print(sum)

sum1=0

i=0

while i<len(x):

#sum1=sum1+x[i] #'set' object does not support indexing

i=i+1

print(sum1)

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

x={10,20,30,40}

y={30,40,50,60}

print(x|y)

print(x.union(y))

print(y|x)

print(y.union(x))

print(x&y)

print(x.intersection(y))

print(y&x)

print(y.intersection(x))

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

Session 21

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

B={4,5,6,7,8}

print(A)

print(B)

print(A-B)

print(A.difference(B))

print(B-A)

print(B.difference(A))

print(A^B)

print(A.symmetric\_difference(B))

print(B^A)

print(B.symmetric\_difference(A))

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

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

print(A)

print(len(A))

A.add(6)

print(A)

print(A.copy())

A.discard(3)

print(A)

print(A.pop())

print(A)

A.remove(2)

print(A)

A.clear()

print(A)

print(len(A))

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

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

print(x)

y=set(x)

print(y)

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

x={p for p in range(10)}

print(x)

y={p\*\*2 for p in range(10)}

print(y)

z={p for p in range(10) if p%2==0}

print(z)

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

noprimes={j for i in range(2,8) for j in range(i\*2,50,i)}

primes=[x for x in range(2,50) if x not in noprimes]

print(noprimes)

print(primes)

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

**4)dict**

Dictionary can be created by using curly brackets {}

OR by calling dict() function

Dictionary represents group of key-value pairs. Each key-value pair in the dictionary is known as an item.

Dictionary is a mutable object.

Keys must be immutable objects.

values can be mutable objects or immutable objects.

Insertion order is not preserved.

Duplicate keys are not allowed but values can be duplicate.

Heterogeneous key and values are allowed.

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

x={}

print(x)

print(type(x))

y=dict()

print(y)

print(type(y))

z={'JAVA':90,'hadoop':85,'python':89}

print(z)

print(type(z))

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

x={'java':90,10:'hyd',True:1234,20:'hyd',30:[1,2,3],'java':95}

print(x)

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

Session 22

x={'hyd':40,'bang':20,'Delhi':15}

print(x)

x['chennai']=42

print(x)

print(x['hyd'])

print(x)

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

x={'hyd':40,'bang':20,'Delhi':15}

print(x)

k=x.keys();

for p in k:

print(p)

v=x.values();

for q in v:

print(q)

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

x='hyd':40

print(x)

i=x.items()

for p in i:

print(p[0],p[1])

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

my\_dict=dict({1:'apple',2:'ball'})

print(my\_dict)

my\_dict1=dict([(1,'apple'),(2,'ball')])

print(my\_dict1)

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

x={'java':90,'python':98,'hadoop':85,'oracle':90}

print(x)

print(x['python'])

print(x.get('hadoop'))

x.pop('oracle')

print(x)

print(x.popitem())

print(x)

x.clear()

print(x)

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

Session 23

marks={}

x=marks.fromkeys(['Math','English','Science'],10)

print(x)

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

marks={}

y=marks.fromkeys(['Math','English','Science'],[10,20,30])

print(y)

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

**Dictionary Comprehension**

x={p:p\*\*2 for p in range(10)}

print(x)

y={q:q\*\*q for q in range(5)}

print(y)

z={r:r\*2 for r in range(10) if r%2==0}

print(z)

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

**Functions:**

test.py

help()

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

sample.py

def <function\_name>(param1,param2, ... ,param\_n):

""" doc string """

statement1

statement2

statement3

[return <value>]

Function is a syntax or structure is used to represent the business logic to perform the operations.

The logic which is represented in function will be executed whenever we call a function

We can call a function for any number times.

def f1():

print("HI")

f1()

f1()

f1()

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

def f1(x,y):

print(x,y)

f1("HI","Student")

f1("Hello","Employee")

The variables which are declared within the function header are known as parameters.

We can define any number of parameters to a function. At the time of calling the function we have to pass the values to the parameters.

The values which are passed to the parameters of a function at the time of calling the function are known as arguments.

Session 24

**return statement:-**

return statement is used to return the execution of the function is over. If we don't assign function call to a variable the return value of a function will become as garbage collectoed. If the programmer doesnot define the return statement explicitly in the function then by default the function returns none.

While executing the function if control reached to the return statement of a function then remaining part of the function will not be executed.

def add(a,b):

c=a+b

return c

x=add(10,20)

print(x)

print add(5,6)

print(add(8,9))

p='python'

y=len(p)

print(y)

add(3,4)

def sub(a,b):

return a-b

z=sub(50,30)

print(z)

def mul(a,b):

d=a\*b

w=mul(50,30)

print(w)

**Types of arguments:-**

We can pass the two types of arguments to the functions.

1)Non-keyword arguments

2)Keyword arguments

**1)Non-keyword arguments:-**

Passing the arguments directly without assigning the arguments to the parameters is known as Non-Keyword arguments.

Non-keyword arguments will work based on the positions.

**2)Keyword arguments:-**

Passing the arguments to the function by assigning the arguments to the parameter names are known as keyword arguments.

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

def greet(name,msg):

print("hello",name,msg)

greet("Python","HI")

greet(msg="Hello",name="Student")

greet("Employee",msg="Good Evening")

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

After keyword arguments we are not allowing to use non-keyword arguments

**Types of parameters:-**

We can define two types of parameters to a function

1)Non-default parameters

2)Default parameters

1)Non-default parameters:-

Defining the parameters without assigning the values explicitly are known as Non-default parameters.

2)Default parameters:-

Defining the parameters by assigning values explicitly are known as default parameters.

We need not pass arguments to the default parameters of a function.

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

def greet(name,msg="HI"):

print("hello",name,msg)

greet("Python")

greet("Student","Good Evening")

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

After defining the default parameters we are not allowed to define non-default parameters. It give syntax error as shown below

def greet(name="student",msg): #error

print("hello",name,msg)

greet("Student","Good Evening")

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

Session 25

def f1(\*x):

print(type(x))

print(x)

print(len(x))

f1()

f1(10,20)

f1(10,20,30)

f1("hyd","Ban","Delhi")

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

If we define the arbitary parameters to a function we can pass 0 or more arguments to that function

Arbitary argument type will be taken as a double.

def f1(\*x):

sum=0

for p in x:

sum=sum+p

print(sum)

f1()

f1(10,20)

f1(10,20,30)

#f1("Hello","HI") #error

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

def f1(dtype,\*x):

if dtype=="int":

sum=0

for p in x:

sum=sum+p

if dtype=="str":

sum=""

for p in x:

sum=sum+p

print(sum)

f1("int",10,20)

f1("int",10,20,30)

f1("str","Hello","HI")

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

def f1(x,y):

print(x)

print(y)

f1(20,y=30)

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

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

**Types of variables:-**

We can define two types of variables in python. They are:

1)global variables

2)local variables

**1)global variables:-**

The variables which are declared outside of the functions are known as global variables.

global variables can be accessed in all the functions. Inorder to modify the global variables data within the functions we need to follow forward declaration mechanism.

**Syntax:-**

global <global\_variable\_name>

**2)local variables:-**

The variables which are declared within the function are known as local variables. Local variables of one function cannot be accessed in other functions.

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

x=0

def f1():

global x

x=x+1

print("in f1")

if x<=5:

f1()

f1()

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

z=300

def f1():

x=100

print(x)

global z

print(z)

z=400

print("in f1")

def f2():

y=200

print(y)

print(z)

print("in f2")

f1()

f2()

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

**Reursive function calling:-**

A function calling itself is called as a recursive function invocation

Infinite recursive function invocation is not supported by Python.

x=0

def f1():

global x

x=x+1

print("Welcome: ",x)

if x<5:

f1()

f1()

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

def fact(x):

if x==1:

return 1

else:

return(x\*fact(x-1))

num=int(input("Enter a number: "))

if num>=1:

print("Factorial of ",num," is: ",fact(num))

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

Session 26

**Anonymous or lambda functions:-**

A function which doesnot contain any name is known as Anonymous function or lambda function.

**Syntax:-**

lambda <parameter>:<expression>

lambda function will not be executed automatically. Inorder to execute the lambda function we have to assign the lambda function to a variable. And we have to call that lambda function through that variable.

p=lambda x:x\*2

print(p(100))

(OR)

def p(x):

return x\*2

print(p(100))

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

def f1(x):

print("In f1")

print(x)

def f2(msg):

return msg

f1(f2("Python"))

f1(f2("Hello"))

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

def f1(p):

print("In f1")

print(p(10))

f1(lambda x:x\*x)

f1(lambda y:y\*y\*y)

Whenever we want to pass the logic as a parameter to the function then we use lambda function.

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

my\_list=[1,5,4,6,8,11,3,12]

new\_list=list(filter(lambda x:(x%2==0),my\_list))

print(new\_list)

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

my\_list=[1,5,4,6,8,11,3,12]

new\_list=list(map(lambda x:x\*2,my\_list))

print(new\_list)

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

input\_list=[1,5,4,6,8,11,3,12]

def my\_map(function,iterable):

output\_list=[]

for p in iterable:

output\_list.append(function(p))

return output\_list

print(my\_map(lambda x:x\*2,input\_list))

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

Session 27

**Modules**

Every Python file itself is known as a Module.

A Python file can contain local variables, functions, classes and executable statements.

We can use the properties of one module to another module by importing that module.

**Normal import:-**

Whenever we import any module by using normal import statement we can access the properties of that module by using the module name only.

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

**mod1.py**

print("From mod1")

x=100

def f1():

print("In f1 of mod1")

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

import mod1

print("Hi from mod2")

y=200

def f2():

print("In f2 of mod2")

print(mod1.x)

print(y)

mod1.f1()

f2()

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

import math

print(dir(math))

print("-------------------")

help(math.sqrt)

print("-------------------")

help(math.ceil)

print("-------------------")

help(len)

print("-------------------")

help(dir)

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

**dir:-**

It is a pre-defined function, which is defined in builtins module.

dir function is used to know the information about given module.

help(dir())

If we don't pass any module name as a parameter to the dir() function then it will provide the information about current module.

**Note:-**

Which Python file directly we are running that Python file technically we call as a main module.

We can know the information about the particular property of particular module by using help() function.

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

import math

p=123

def m1():

print("HI")

print(dir())

print("---------------")

print(dir(math))

print("---------------")

print(math.sqrt)

print("---------------")

print(math.ceil)

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

dir() and help() are important. Because we use these functions regularly

**Renaming a module:-**

Instead of accessing the properties of a module always by using module name whenever we use normal import statement we can create alias name for a module and access the properties of that module by using alias name.

After creating the alias name or rename for a module we are not allowed to access the properties of that module by using its original name.

import math as a

import mod1 as b

print("-------")

print(a.pi)

print(b.x)

print(a.sqrt(9))

b.f1()

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

from math import pi,sqrt

from mod1 import x,f1

z=3000

def f3():

print("In f3")

print(pi)

print(x)

print(z)

print(sqrt(9))

f1()

f3()

print(dir())

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

Session 28

from mod1 import \*

from math import \*

y=3000

def f2():

print("In f2 of test")

print(x)

print(pi)

print(y)

f1()

print(sqrt(9))

f2()

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

from mod1 import x,p1

from math import pi,sqrt

y=3000

def f2():

print("In f2 of test")

print(x)

print(pi)

print(y)

f1()

print(sqrt(9))

f2()

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

**from import with '\*'**

It is used to make it available all the properties of one module into another module.

It is never recomended to use the '\*' import if there is no specific requirement.

import mod1

from math import \*

y=3000

def f2():

print("In f2 of test")

#print(x)

print(pi)

print(y)

#f1()

print(sqrt(9))

f2()

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

**builtins module**

builtins is a pre-defined module, which contains frequently used logics.

In every Python module directly we can access the properties of builtins module, because Python interpretor automatically imports the builtins module into the every Python file.

from builtins import \*

x='Python'

print(len(x))

help(len)

The above program will work in Python3.8 version

Use \_\_builtin\_\_ to work in Python2.7 as shown below.

from \_\_builtin\_\_ import \*

x='Python'

print(len(x))

help(len)

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

import os

cwd1=os.getcwd()

print("1",cwd1)

os.chdir("hw")

cwd2=os.getcwd()

print("2",cwd2)

os.chdir(os.pardir)

cwd3=os.getcwd()

print("3",cwd3)

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

Session 29

import os

if os.name=='nt':

command="ipconfig"

else:

command="pwd"

print(command)

**Reloading a module**

Whenever we import a module for more than once then by default that module is going to be loaded into the memory only once.

Inorder to load the module into the memory location once again we use reload function of **imp** module.

mod1.py

print("hi")

x=100

def f1():

print("In f1 of mod1")

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

import mod1

import imp

print(mod1.x)

mod1.f1()

imp.reload(mod1)

print("bye")

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

**Modules Search Path**

Whenever we import a module then Python interpretor searches for the imported modules in the following order.

1) Current/Present working directory(main module location)

2) Python path(Environment variable)

3)Installation dependent default directory

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

Session 30

**Packages**

Packages nothing but a folder or a directory which represents collection of python modules.

A Python package can contain another packages called sub packages.

We can import the properties of modules of packages using

import <package\_name>.<module\_name>

from <package\_name>.<module\_name> import <property>

create directory with d1 and place below x.py in d1

x.py

a=10

def f1():

print(“In f1 of x”)

create directory with d2 in d1 and place below y.py in d2

y.py

b=20

def f2():

print(“In f2 of y”)

test.py

import d1.x

from d1.d2.y import b,f2

print(d1.x.a)

d1.x.f1()

print(b)

f2()

This test.py must be outside d1 folder.

**Exception Handling**

Generally we will get the two types of errors in any programming Language. They are:

1) Syntax Error

2) Runtime Error

**1) Syntax Error**

The errors which occurs because of syntax mistakes are known as syntax errors.

If any syntax errors are there in our Python program then program execution will not be started.

Python programmers and developers are responssible for providing the solutions to the syntax error.

The errors which occurs after starting the execution of the program are known as runtime errors.

Generally we will get the runtime errors because of

a) invalid data.

b)Program logic

c) Memory related issues

d) Hardware failures etc

With respect to every runtime error corresponding runtime error representation class is available.

Runtime error representation classes technically we call as Exception classes.

While executing the program if any runtime error is occured then automatically corresponding runtime error representation class object will be created. Creating runtime error representation class object is technically known as a raising exception.

If the Python program doesnot contain any code to handle the raised exeption then program is going to terminate abnormally.

**Abnormal Termination**

Concept of terminating the program in the middle of its execution without executing the last statement of the program is known as abnormal termination. Abnormal termination is the undesirable situation in any programming Language.

Session 31

print("begin")

x=int(input("enter fno: "))

y=int(input("enter sno: "))

x=x/y

print(x)

print("end")

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

**Exception Handling:**

The concept of identifying which runtime error representation class object is created, receiving that object and assigning that object to corresponding runtime error representation class is known as Exception Handling.

We can implement exception handling in Python by using try and except blocks.

**try block:-**

A block which is preceded by the try keyword is known as a try block.

try block syntax

try:

statement1

statement2

.

.

.

statement\_n

The statements which causes to runtime errors and other statements which depends on the execution of runtime error occured statements are recomended to represent in try block

While executing the try block if any runtime error is occured then immediately that runtime error represented class object is identified, received by the try block and forward that object to except block without executing remaining statements of the try block

**except block:**

A block which is preceded by except keyword is known as except block.

Default Except block syntax:-

except:

statement1

statement2

.

.

.

statement\_n

Named Except block syntax:-

except(RuntimeErrorRepresentationClass):

statement1

statement2

.

.

.

statement\_n

Except block receives the runtime error representation class objects which is given by the try block and assign that object to the corresponding runtime error representation class.

While executing the try block statements if exception is raised then only control will go to the except block

In except block we can define the statements to display the user friendly error messages

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

print("begin")

x=int(input("enter fno: "))

y=int(input("enter sno: "))

try:

z=x/y

print(z)

except(ZeroDivisionError):

print("sno cannot be zero")

print("end")

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

Default Except block can handle any type of Exception.

print("begin")

x=int(input("enter fno: "))

y=int(input("enter sno: "))

try:

z=x/y

print(z)

except:

print("sno cannot be zero")

print("end")

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

Session 32

**Simple try with multiple except blocks:**

It is recomended to handle the different exceptions by using the different except blocks to display the corresponding user friendly error messages.

Whenever we define single try with multiple except blocks, if exception is raised in the try block then control will go to first except block.

If first except block is not handled that exception then control will go to second except block and so on.

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

print("begin")

x=int(input("enter fno: "))

y=int(input("enter sno: "))

try:

z=x/y

print(z)

except(ValueError):

print("enter number is value only")

except(ZeroDivisionError):

print("sno cannot be zero")

except:

print("error occured")

print("end")

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

Default except block must be the last block otherwise it gives error.

Note:-

Place default block as first or second block and check the result.

syntax error

Default except block should be last.

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

print("begin")

x=int(input("enter fno: "))

y=int(input("enter sno: "))

try:

z=x/y

print(z)

except:

print("error occured")

except(ValueError):

print("enter number is value only")

except(ZeroDivisionError):

print("sno cannot be zero")

print("end")

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

**finally block:-**

A block which is preceded by the finally keyword is known as finally block.

**syntax:**

finally:

statement1

statement2

.

.

.

statement\_n

The set of statements which must be execute whether exception is raised or not raised, eventhough exception is raised whether it is handled or not handled are recomended to write in finally block.

Resource releasing statements(file closing statements, database connection closing statements) are recomended to write in finally bloak.

try:

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

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

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

finally:

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

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

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

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

try:

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

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

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

except:

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

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

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

finally:

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

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

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

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

print("begin")

x=int(input("enter fno: "))

y=int(input("enter sno: "))

try:

z=x/y

print(z)

except(ZeroDivisionError):

print("sno cannot be zero")

finally:

print("in finally block")

print("end")

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

**Nested try-except-finally block:**

We can define the one try block inside another try block.

A try block which contains another try block is known as a outer block.

A try block which is defined in another try block is known as inner try block.

If exception is raised in the outer try block that can be handled only by using outer try except block.

If exception is raised in the inner try block that can be handled using inner try-except block, if inner try-except block is not handled then we can also handle using outer try-except block.

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

try:

print("In outer try")

try:

print("In inner try")

except:

print("In inner except")

finally:

print("In inner finally")

except:

print("In outer except")

try:

print("In try of except")

except:

print("In except of except")

finally:

print("In finally of except")

finally:

print("In outer finally")

try:

print("In try of finally")

except:

print("In except of finally")

finally:

print("In finally of finally")

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

Session 33

**User defined Exceptions:**

The exception classes which are defined by the programers according to their business requirements are known as user defined Exceptions.

**Steps to implement user defined Exceptions:-**

1)Defining user defined Exception class:-

Any user defined class which is extending any one of the pre-defined exception class is known as a user defined Exception class

Syntax

class <class\_name>(Exception):

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

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

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

2)Raising the user defined Exception explicitly:-

Creating the exception class objects is technically known as a raising the Exception.

Pre-defined Exceptions raised automatically whenever corresponding runtime error is occured.

We have to raise user defined exceptions explicitly according to our business requirements by using following syntax.

Syntax:

raise <Exception\_class\_name>

3)Handling the raised Exceptions:

We can handle the raised Exceptions by using try and except blocks

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

class Error(Exception):

"""Base class for other Exceptions"""

pass

class ValueTooSmallError(Error):

"""Raised when the value is too small"""

pass

class ValueTooLargeError(Error):

"""Raised when the value is too long"""

pass

num=10

while True:

try:

i\_num=int(input("Enter a number: "))

if i\_num<num:

raise ValueTooSmallError

elif i\_num>num:

raise ValueTooLargeError

break

except ValueTooSmallError:

print("Too small, Try again")

except ValueTooLargeError:

print("Too long, Try again")

print()

print("Your guess is correct")

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

**File Handling:-**

Programming Languages are good at processing the data. But they cannot store the data permanantly. Because programming Languages programs memory will be allocated in RAM.

RAM is a volatile so that once the program execution is over the memory which is allocated for that program is going to be de-allocated.

If we don't store the data permanantly we cannot able to store the data in future.

Inorder to store the data permanantly we use Files, RDBMS or ORDBMS(Object relational database management system), NOSQL

File is a named location on the disk to store related information. Python Language is providing API or Libraries to perform the operations on the files.

Before going to perform the read|write operations on the files first we have to open the files. We can open the files by calling the open function of builtins module.

At the time of opening the file we need to specify the mode of the file

**Mode**  **Description**

'r' Opens the file for reading (default)

'w' Opens the file in writing. Creates a new file if it does not exist or truncate the file if it exists.

'x' Opens the file in exclusive creation. If the file already exists, the operation fails.

'a' Open for appending at the end of the file. Creates a new file of it does not exists.

't' Opens in text mode(Default)

'b' Opens in binary mode.

'+' Opens file for updating(Reading and Writing)

After executing the open function it returns file object.

File Object provides some methods to read or write the data to the files. After performing the operations on the file we have to close the file. We can close the file by calling close() method on the file Object.

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

test.txt

First Line

Second Line

<b>Third Line</b>

Fourth Line

<i>Sixth Line</i>

Last Line

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

**Reading data from the file**

one.py

x=open("test.txt")

print(x.read())

x.close()

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

x=None

try:

x=open("z.txt")

print("File opened")

print(x.write("Python"))

except:

print("Error occured")

finally:

if x!=None:

x.close()

print("File Closed")

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

Session 34

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

**Writing data to file**

x=open("z.txt","w")

x.write("Welcome")

x.close()

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

**Appending data to file**

x=open("z.txt","a")

x.write("Python")

x.close()

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

**Working with file methods**

x=open("z.txt")

print(x.tell())

print(x.read(3))

print(x.tell())

print(x.read(4))

print(x.tell())

x.seek(5)

print(x.tell())

print(x.read(10))

print(x.tell())

x.close()

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