

PYTHON

By

Mr. Lokesh



SRI SAI BHAVANI XEROX

Ram Mirra Towers, Ground Floor
Beside Satyam Theatre
Ameerpet, HYD
Cell: 99 08 19 59 58

Copyright © 2006 by Pearson Education, Inc., or its affiliates. All Rights Reserved.

Python

Applications of python

| | Java | .net |
|------------------|------|----------------------------|
| * Automation app | X | X |
| * Data Analytics | X | X |
| * Scientific app | X | X |
| * Web app | ✓ | ✓ |
| * Web scrapping | X | X |
| * Test cases | ✓ | ✓ |
| * N/w with IOT | ✓ | X |
| * Admin script | X | X |
| * GUI | ✓ | ✓ Graphical User Interface |
| * Gaming | ✓ | ✓ |
| * Animation | X | X |

Addition of 2 no's :-

Program:-

```
Print (input ("enter fno") + input ("enter sno"))
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Note:- N/w is possible in Java but not with IOT

IOT - Internet of things.

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

① Automation

- * They are using python in Robotics.
- * Robotics are electronic device. Generally robotics don't have intelligence. By Applying Artificial intelligence algorithms. We apply A.I by using python.

② Data analytics :-

- * Languages are for processing the data but not for storing the data.
- * Data can be stored in
 - i) files
 - ii) ORDBMS
 - iii) NOSQL
 - iv) HADOOP

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

After reading the data, the data is represented in

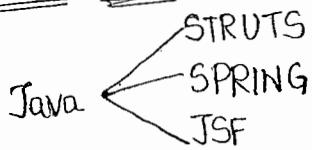
pie-chart, Bar graph, curve lines for visualization.

- * 'R' language is used for analytics but not used for streaming

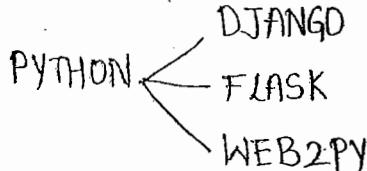
③ Web app:-

- * Java, PHP, .net and also python used for web applications.
- * Web sites cannot be developed directly using languages.
- * Websites are developed by using frameworks.

Frame Works



.NET → MVC



SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

④ Scientific applications:-

- * NASA is using only a python language to develop scientific applications.

⑤ Web Scrapping:-

- * Now-a-days Google, Yahoo!, YouTube are using web scrapping.
For example:- If you are searching for iPhone 7S in google. The next day if you are opening e-commerce website i.e., Amazon there you can see the products of iPhone 7S. This is because of web scrapping.

⑥ IoT with IOT:-

- * IoT allows you to connect to various IoT's. It can also be connected to the smartphones, washing machines etc..
- * IoT cyber attacks (Hackers) are done through IoT devices.
- * Because IoT was not having much security it means they are not secured upto the mark.

⑦ Admin Script:-

- * It is used for higher level administration

Programming languages:-

- 1) procedure oriented programming language:- C → functions.
 - 2) Object Oriented programming language:- C++, JAVA → OOPS
 - 3) Scripting language:- S-S → interpretation
 - 4) Module programming language:- modula-2, modula-3 → modules.
- * Python supports the four, procedure, object, scripting and module programming language.
 - * It is suitable for gaming and hardware drivers.

- * python was implemented by Guido van Rossum in 1989
- * Python is called General purpose language.
- * In 1991 G.V.R implemented python and made it available to public
- * An open community of python is a non profitable company & python software foundation has taken as a third party of python by Guido van Rossum
- * Java author was sun micro system. Java is open source
- * .net is of microsoft.
- * In python language the pre defined programs are called modules or packages.
- * 1 lakh 7000+ modules are available in python currently
- * 2.7 test version of python software available (old)
- * 3.6 test version of python software available (new)
- * Python is platform independent.
- * Guido van Rossum implemented python language by taking different varieties of languages speeches like

- ① procedure oriented programming language
- ② object oriented programming language
- ③ Scripting programming language
- ④ Modular programming language

→ By using python language we can implement different variety of applications like

- 1) Automation apps.
- 2) Data Analytics
- 3) Web apps
- 4) Scientific obligations

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Raja Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

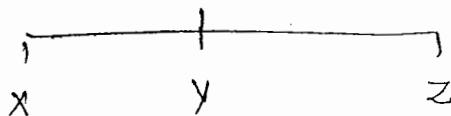
- 5) Web scrapping techniques
- 6) NW with IoT applications
- 7) Administration script
- 8) Test cases
- 9) GUI applications
- 10) Gaming applications
- 11) Animation applications

SRI SAI BHAVANI XEROX
 H.No. 7-1-209/1, Ground Floor,
 Ram Mira Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

* gyro vendrosom made it available python language to the public from 1981

- * Currently copy rights of the python language are registered with an open community and non profitable organization called as python software foundation
- * python language is open source and also source code is also open source
- * Java language is open source but source is not open source
- * .Net is not an open source

P.S.F (Cpython)



distribution
python.

SRI SAI BHAVANI XEROX
 H.No. 7-1-209/1, Ground Floor,
 Ram Mira Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

* The python software which is developed by the python software foundation is known as 'cpython'.

* There are so many distributions are available for Cpython.

* Cpython is an open source and platform independent.

* The current version of the Cpython is 3.6.1

* With respect to every operating system separate python softwares are available for with respect to every version of python.

* download the required 3.6.1 python executable file from the www.python.org.com website.

Installation :-

* Click on the downloaded executable installer

* Click on 'Run'

* Click on 'customize installation'.

* Click on 'Next'

* Select the installation location by clicking on browse button

[c:/python 35-32].

* Click on 'install'

* Click on 'Yes'

* Click on 'close'.

Path:- path is an environmental variable of operating system by using which we can make it available the softwares which are installed in one directory to all other directories of the operating system.

To set path:-

Right click on my computer



click on properties



click on advanced system setting



click on advanced



click on environment variables.



Go to system variables, select 'path'



Click on 'edit'



copy the installation folder location of python

software in the beginning of the variable value [c:]python 35-35



click on 'OK'



click on 'OK'



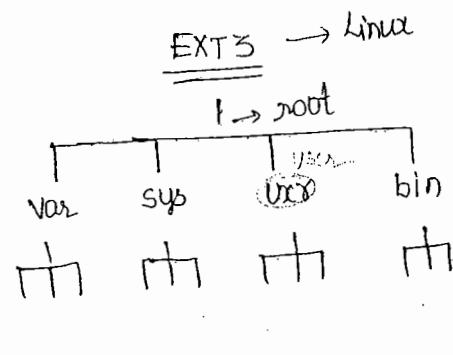
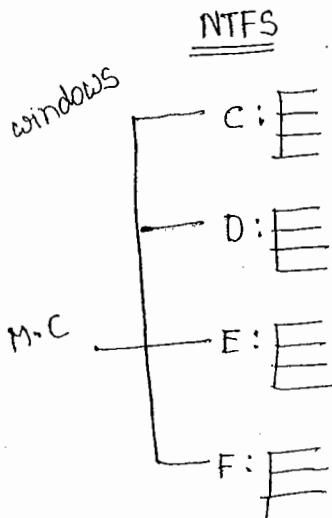
click on 'OK'

Now path setting is secured.

* python programs can be developed in two modes.

1) Interactive mode

2) Batch mode.



SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Niya Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* Windows following file format NTFS file system

Linux following file format EXT3 file system

User → R, W, X

R → Read

Group → R, W, X

W → Write

Other → R, W, X

X → Execute

Implementation of python program

Python program can be implemented by two ways:

1) Interactive mode (submit statement by statement explicitly)

2) Batch mode (writing all statements and submit all statements)

↓
editors and IDE's are available in the batch mode

IDE - Integrated development Environment

* In interactive mode python command shell is required. It is available in installation of python cell.

* The interactive mode is not suitable for developing the projects & applications.

* Interactive mode is used for predefined function and programs.

Ex:-

>>> x = 1000

>>> y = 2000

>>> x+y

3000

>>> quit

>>> x+y

>>> x,y is not find.

* Interactive mode is unfit for looping purpose.

SRI SAI BHAVANI XTPX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Interactive mode:-

- * The concept of submitting one by one python statements explicitly in the python interpreter is known as "interactive mode"
- * In order to submit the one by one python statements explicitly to the python interpreter we use python command line shell.
- * Python command line shell is present in python software
- * We can open the python command line shell by executing python command on command prompt or terminal.

Ex:- c:\users\lokesh > python

```
>>> 3+4  
7  
>>> 'sathya'*3  
'sathya sathya sathya'  
>>> x = 1000  
>>> y = 2000  
>>> x+y  
3000  
>>> quit  
>>> x+y  
c:\users\lokesh > python
```

Error: name 'x' not defined.

Implemented in Linux by using Interactive mode:-

[training @ local host] \$: python

remaining same as python.

The only difference between python and linux is 'b'.
use doesn't use quit() in this program we use .

- * Whenever we exit from python command line shell environment the work which we have done on python command line shell environment is going to lose.
- * By Interactive mode is not suitable for developing the applications and projects.
- * Interactive mode is suitable for leaving python and to test predefined functions and functionality.

Batch Mode:-

In the concept of writing the group of python statements in a file, save the file with extension .py and submit that entire file to the python interpreter is known as batchmode.

- * Inorder to develop the python files we use editors or IDE's
- * Different editors are notepad, notepad ++, edit +, vi, nano, gedit and so on.
- * Open the notepad and write the following code.

Ex:-

```
x=1000
y=2000
print(x+y, x-y, x*y)
```

- * Save the file in D drive qam python folder with the demo.py.

- * Open command prompt and execute following commands.
python o:\qam python \demo.py

```
3000
-1000
2000.000
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* Save another method if we correctly set the path

D:

D:\>d qam python

D:\l am python > python Demo.py

3000

-1000

2000.000

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Problems by using Editors:-

- * By using editors, time consumption is more & vice versa
cost also increases.
- * If we develop the python programs or applications by using editors then we face following problems.
 - 1) Development time and cost will be increased.
 - 2) Automatic debugging operations can't be applied.
 - 3) Code generation tools can't be used.

To overcome the above problems we develop the python applications by using Integrated development Environment (IDE's)

Different IDE's are

Phycharm, ERIC, ECLIPSE, NETBEANS

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Download and Install the phycharm community IDE :-

* Open the phycharm IDE by clicking on shortcut icon.

* Click on file



Click on new project



Enter the project location as 'qam sample proj'



Click on create

* Right click on qamsample proj



Click on 'new'



click on python file **SRI SAI BHAVANI XEROX**
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016



Enter the name as 'demo'



Click on 'OK'

Write the following program in demo.py.

Program:- $x=100$

$y=200$

`print(x+y)`

`print(x-y)`

`print(x * y)`

→ Click on 'Run'

→ Click on 'Run'

→ Click on 'Demo'

Op:- 300

-100

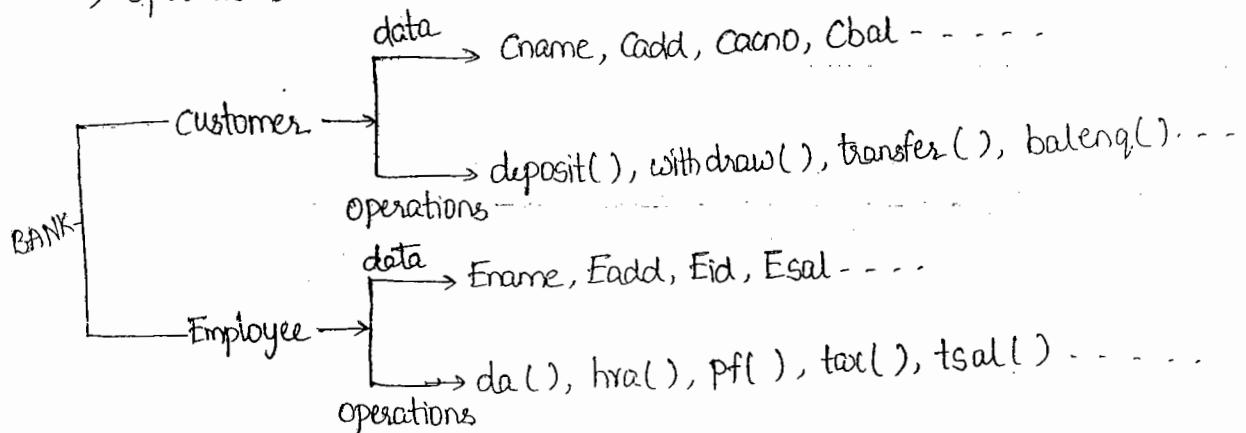
20,000

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Business Applications:-

* Any business organisation use case or scenario contains two parts they are

- 1) data
- 2) operations.



* Data of organisation usecases can be represented in any programming language by using datatypes and variables.

* Every programming languages provides datatypes and variable but the datatypes and variables are one programming language are not going to be same with the datatypes and variables of the another programming language.

* Every programming languages provides functions/ methods or both to represent the operations.

Data types:-

Datatypes are not nothing but some of the keywords of the programming language, which are used to specify what type of data has to be stored into the variables.

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* Programming languages supports two different varieties of data types:

- 1) static data types
- 2) Dynamic data types.

Static Data types:-

In static data supported languages programmer should define the data type to the variable explicitly.

* In static data types supported languages one variable can store one variety of data.

* C, C++, JAVA, .NET languages are supporting static data types.

Dynamic Data types:-

* In dynamic data types supported languages programmer should not define the data type to variable explicitly.

* At the time of execution of the program based on the data which is assigned to the variable, data type of the variable is decided.

* In dynamic data types supported languages one variable can store different varieties of data.

* Python, JavaScript . . . languages are supporting dynamic data types.

Python language supports following data types.

- 1) int
- 2) float
- 3) complex
- 4) bool
- 5) str
- 6) list
- 7) tuple
- 8) set
- 9) dict

* Every data type in python language is internally implemented as a class.

* Python language data types are categorized into two types.

They are

1) Fundamental types.

2) Collection types.

Fundamental Types:-

* Fundamental types represented classes, objects, are storing only one value

* Python supports following fundamental types

int

float

complex

bool

str

Program:-

a = 1000

print(a)

print(type(a)) int

print(id(a))

b = 123.123

print(b)

print(type(b)) float

print(id(b))

c = 3+4j

print(c)

print(type(c))

print(id(c))

type(l)

print(l)

id(l)

↓

address

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
d=True  
print(d)  
print(type(d))  
print(id(d))  
e=sathya  
print(e)  
print(type(e))  
print(id(e))
```

Output:-

1000

<class 'int'>

36785936

123.123

<class 'float'>

36786160

(3+4j)

<class 'complex'>

36654496

True

<class 'bool'>

1809368336

sathya

<class 'str'>

38668064

* Python supports two types of objects. They are

- 1) Immutable objects
- 2) Mutable objects.

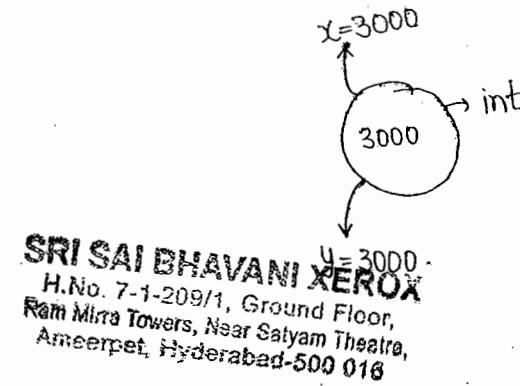
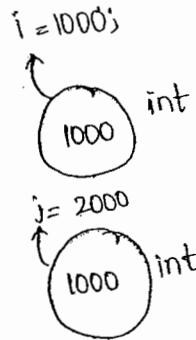
SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

1) Immutable Objects:-

The objects which doesn't allow to modify the contents of those objects are known as "Immutable objects."

- * Before creating immutable objects with some content python interpreter verifies is already any object is available in memory location with same content or not.
- * If already object is not available then python interpreter creates new objects with that content and store that object address into reference variable.
- * If already object is present in memory location with the same content without creating new object already existing object address will be given to the reference variable.



Program:-

```
i = 1000
print(i)
print(type(i))
print(id(i))

j = 2000
print(j)
print(type(j))
print(id(j))
```

```

x = 3000
print(x)
print(type(x))
print(id(x))
y = 3000
print(y)
print(type(y))
print(id(y))

```

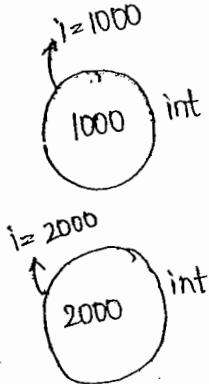
SRI SAI BHAVANI XEROX
 H. No. 7-1-209/1, Ground Floor,
 Ram Mira Towers, Near Satyam Theatre,
 Amarpet, Hyderabad-500 016

O/P:-

```

1000
<class 'int'>
36785936
2000
<class 'int'>
37520144
3000
<class 'int'>
37520192
3000
<class 'int'>
37520192
>>>
>>>

```



SRI SAI BHAVANI XEROX
 H. No. 7-1-209/1, Ground Floor,
 Ram Mira Towers, Near Satyam Theatre,
 Amarpet, Hyderabad-500 016

Here already existing one is deleted and creating new object and address also different so 'int' is immutable object

* Memory space can be saved in immutable objects.

Program:-

```
i = 1000
print(i)
print(type(i))
print(id(i))
i = 2000
print(i)
print(type(i))
print(id(i))
```

O/P:-

```
1000
<class 'int'>
36785936
2000
<class 'int'>
37519728
>>>
```

- * int, float, complex, bool, str, tuple are immutable objects.
- * Immutable objects performance is high
- * Applying Iterations on immutable objects takes less time.
- * All fundamental types represented classes objects and tuple class objects are immutable objects.

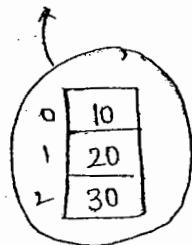
Mutable objects:-

- The objects which allows to modify the contents of those objects are known as 'Mutable objects'.

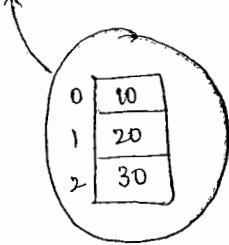
- * We can create two different mutable objects with same content.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

$x = [10, 20, 30]$



$y = [10, 20, 30]$



program:-

$x = [10, 20, 30]$

print(x)

print(type(x))

print(id(x))

$y = [10, 20, 30]$

print(y)

print(type(y))

print(id(y))

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyan Theatre,
Ameerpet, Hyderabad-500 016

Output:- [10, 20, 30]

<class 'list'>

37561608

[10, 20, 30]

<class 'list'>

37117704

>>>

>>>

program:- $x = [10, 20, 30]$

print(x)

print(type(x))

print(id(x))

$x[1] = 123$

print(x)

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyan Theatre,
Ameerpet, Hyderabad-500 016

print(type(x))

print(id(x))

Output: [10, 20, 30]

<class 'list'>

37565624

[10, 20, 30] [10, 123, 30]

<class 'list'>

37565624

SRI SAI BHAVANI XE

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * list, set, dict classes objects are mutable objects
- * Mutable objects performance is low when compared to immutable objects.
- * Applying Iterations on mutable objects takes huge time.

String Handling:-

- * Group of characters or sequence of characters is known as 'string'
- * In python strings can be stored in 'str' class objects.
- * 'str' class objects can be created in two ways.
 - 1) single codes ' ' , " " or u " "
 - 2) triple codes """ , "" " or " " " " " "
- * Single codes is used to represent one line string
- * Triple codes is used to represent multiple line string.
- * In str class objects every character is represented with unique index.

- * 'str' objects are supporting both positive indexing and negative indexing.
- * Positive indexing starts from zero '0' and negative indexing starts from '-1'
- * By using indices we can read the data from the str objects.

Program:-

```
x= 'sathya technologies'
```

```
print(x)
```

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

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

```
y= " ameerpet,
```

```
hyderabad
```

```
telangana" "
```

```
print(y)
```

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

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

Output:-

```
sathya technologies
```

```
<class 'str'>
```

```
6099792
```

```
ameerpet
```

```
hyderabad
```

```
telangana
```

```
<class 'str'>
```

```
6080456
```

```
>>>
```

SRI SAI BHAVAN XEROX
H.No. T-1-209/1, Ground Floor,
New Alfa Towers, Near Satyam Theatre,
Ameerpet, Hyderabad - 500 016

Index & Slicing method

Ex:-

| | | |
|---|---|-----|
| 0 | S | -10 |
| 1 | a | -9 |
| 2 | t | -8 |
| 3 | h | -7 |
| 4 | y | -6 |
| 5 | a | -5 |
| 6 | t | -4 |
| 7 | e | -3 |
| 8 | c | -2 |
| 9 | b | -1 |

Program:-

$x = \text{'Sathyatech'}$

print(x) - (1) sathyatech

print($x[2]$) - (2) t

print($x[-2]$) - (3)

print($x[1:5]$) - (4) saty
 ↑ inclusive

print($x[1:5]$) - (5) saty
 ↑ exclusive

print($x[5:-1]$) - (6) tech

print($x[:7]$) - (7) sathyat

print($x[4:1]$) - (8) no

print($x[-6:-1]$) - (9) yatec

print($x[-2:-8]$) - (10) no

print($x[2:-2]$) - (11) thyati

print($x[8:-7]$) - (12) no

SRI SAI BHAVAN
H.No. 7-1-208/1, Ground Floor,
Ram Niwas Towers, Nizam Salwar,
Ameerpet, Hyderabad, Telangana, India

Output:-

sathyatech

t

c

athy

attech

Sathyat

yatec

thyate

>>>

str } immutable objects

list } mutable object

tuple } Iterable Objects

set } immutable object

dict } mutable object

* Iterable objects, has predefined function len()

* len() built in function is used to find the number of characters.

Eg:- x = 'sathyatech'

print(x)

print(len(x))

y = 1000

print(y)

print(len(y))

Op:- sathyatech

10

1000

len(y) is not provided because int() has no len(). As int() is not an Iterable object.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Sayaji Theatre,
Amarpet, Hyderabad-500 015

Reading the data from keyboard:-

- * We can read the data from the keyboard by calling `input()` function
- * Input function is predefined function which reads the data in the form of string format only
- * After reading the data in the form of string format we can convert string represented data in the form of required form by using type conversion function.

Ex:- `fn = input("enter fname")
ln = input("enter lname")
print(fn+ln)`

O/p:-
enter fname sathya
enter lname tech
sathyatech
>>>

program:-
`i = input("enter fno")
j = input("enter sno")
print(i+j)`

O/p:-
enter fno 1000
enter sno 2000
10002000
>>>

SRI SAI BHAVAN
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Amarpet, Hyderabad-500 016

program:-

```
i= input ("enter fno")  
x= int(i)  
j= input ("enter sno")  
y= int(j)  
print (x+y)
```

O/p:- enter fno 2000
enter sno 1000
3000
>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

program:-

```
a= input ("enter int value")  
print (type(a))  
b= int (a)  
print (type(b))  
c= input ("enter float value")  
print (type(c))  
d= float (c)  
print (type(d))  
P= input ("enter complex value")  
print (type(p))  
q= complex (p)  
print (type(q))  
x= input ("enter bool value")  
print (type(x))  
y= bool(x)  
print (type(y))
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Op:- enter int value 1000
<class 'str'>
<class 'int'>
enter float value 123.123
<class 'str'>
<class 'float'>
enter complex value 3+4j
<class 'str'>
<class 'complex'>
enter bool value True
<class 'str'>
<class 'bool'>
>>>

program:-

```
a = "sathya"  
b = "tech"  
c = 1000  
d = 2000  
e = "1234" ... This is string  
print(a+b)  
print(c+d)  
print(a+e)  
print(a+str(c)) → We have not converted string into int so we change int → string  
print(c+int(e)) → Here we can modify either 'c' → string (or) e → int  
print(str(c)+e) ... Both are possible
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Op:- sathyatech

3000
Sathya1234
Sathya1000
2234
10001234
→→→

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

program:-

```
a = "sathya"  
print (id (a))  
print (a)  
print (len(a))  
if a[2] = 'p'  
    b = "tech"  
    print (id (b))  
    print (b)  
a = a+b  
print (a)  
print (id (a))
```

O/P:- 37599200

sathya

6

37599328

tech

sathyatech

37594680

>>>

Operators:-

* Operators are the constructs which are used to perform the operations on the data of the objects which are pointed by operands

(or)

* To perform the operations on the addresses of the objects which are pointed by operands.

* python supports following types of operators.

- i) Arithmetic operators
- ii) Comparison (Relational operators)
- iii) Logical (Boolean operators)
- iv) Bitwise operators
- v) Assignment operators
- vi) Special operators.

Arithmetic operators:-

* Arithmetic operators are used to perform mathematical arithmetic operations like addition, subtraction, multiplication, division, floor division, modulus and exponential.

$$x=15$$

$$y=4$$

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

```
print ('x-y = ', x-y)
```

```
print ('x*y = ', x*y)
```

```
print ('x/y = ', x/y)
```

```
print ('x//y = ', x//y)
```

```
print ('x % y = ', x%y)
```

```
print ('x ** y = ', x**y)
```

O/P:-

$$x+y=19$$

$$x-y=11$$

$$x*y=60$$

$$x/y=3.75$$

$x//y=3$ (before the point value)

$x \% y=3$ (it gives the remainder)

$x ** y = 537824 (14^5)$

SRI SAI BHAVAN
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam IT Park,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVAN
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam IT Park,
Ameerpet, Hyderabad-500 016

Comparison Operators:-

Comparison operators are used to compare the data of

the objects which are pointed by the operators.

Ex:- $x=10$

$y=12$

point ($x < y$ is , $x < y$)

point ($x > y$ is , $x > y$)

point ($x == y$ is , $x == y$)

point ($x != y$ is , $x != y$)

point ($x <= y$ is , $x <= y$)

point ($x >= y$ is , $x >= y$)

$x < y$ is true

$x > y$ is false

$x == y$ is true

$x != y$ is false

$x <= y$ is true

$x >= y$ is false

$<>$

$x < y$ is true

mathematical logical operations

logical operators are used to perform the

Logical Operators:-

$x = True$

$y = False$

point ("not x is , not x")

point ("x or y is , x or y")

point ("x and y is , x and y")

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mills Towers, Near System Theatre,
Ameerpet, Hyderabad-500 016

OP:- x and y is false

x or y is true

not x is false.

Bitwise Operators:-

Bitwise operators converts the data in the form of binary format and performs the operations on the binary data

→ bitwise operator gives the results in the form of decimal format

Ex:- $x = 10$

$y = 4$

`print(x & y)` → Bitwise and

`print(x | y)` → Bitwise or

`print(x ^ y)` → Bitwise XOR

`print(~x)` → Bitwise NOT

`print(x >> 2)` → Bitwise rightshift

`print(x << 2)` → Bitwise leftshift

OP:-

0

2.5

14

-11

2

40

>>>

Assignment operators:-

Assignment operators are used to assign the data to the variables.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Eg:-

| operator | Example | Equivalent to |
|----------|---------|---------------|
| = | $x=5$ | $x=5$ |
| += | $x+=5$ | $x=x+5$ |
| -= | $x-=5$ | $x=x-5$ |
| *= | $x*=5$ | $x=x*5$ |
| /= | $x/=5$ | $x=x/5$ |
| %= | $x\%=5$ | $x=x \% 5$ |
| //= | $x//5$ | $x=x//5$ |
| **= | $x**=5$ | $x=x**5$ |
| &= | $x&=5$ | $x=x&5$ |
| = | $x =5$ | $x=x 5$ |
| ^= | $x^=5$ | $x=x^5$ |
| >>= | $x>>=5$ | $x=x>>5$ |
| <<= | $x<<=5$ | $x=x<<5$ |

Special Operators:-

python supports two types of special operators. They are

- 1) Identity operator
- 2) membership operator.

1) Identity operator:-

Identity operators are used to compare the addresses of the objects, which are pointed by the operands.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

program:-

i=1000

print(i)

print(id(i))

j=2000

print(j)

print(id(j))

print(i is j)

print(i is not j)

K=3000

print(K)

print(id(K))

x=3000

print(x)

print(id(x))

print(K is x)

print(K is not x)

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)

O/P:- 1000

36790032

2000

37523808

False

True

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

3000

37523824

3000

37523824

True

False

[10, 20, 30]

37125176

[10, 20, 30]

37124576

False

True

>>>

2) Membership Operator:-

Membership operators are used to search for the required element in the given iterable object.

Program:- i = 'satya'

```
print ('t' in i)
```

```
print ('t' not in i)
```

```
print ('p' in i)
```

```
print ('p' not in i)
```

Op:-

True

False

False

True

>>>

Program:- i = 1000

```
print ('t' in i)
```

Op:-

Error

int is not iterable object

Control flow statements:-

By default python program execution starts from 1st line, executes each and every statement only once and terminates the program if the last statement of the program execution is over.

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

Condition:-

- * After evaluating the expression result is given as boolean value then we call that expression as a condition.
- * Every condition is a expression, but every expression is not a condition.

Blocks:-

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

1. Blocks begin when the indentation increases.

2. Blocks can contain other blocks.

3. Blocks end when the indentation decreases to zero or to end its containing block's indentation.



Python supports two types of control flow statements:

- 1) conditional statement
- 2) looping statement

① Conditional statement:- Conditional statements are used to decide whether block as to execute or skip the execution of the block.

python supports three conditional statements. They are

1) if

2) else

3) elif

4) If:-

syntax:-

if condition : statement

(or)

if condition

stmt 1

stmt 2

condition returns true it executes the block

otherwise skip the execution of the block.

program:-

print ("begin")

x=input ("enter positive no")

i=int(x)

if i<10:

print("given no is 1 digit no")

print ("end")

O/P:-

begin

enter positive no5

given no is 1 digit no

end

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

begin
enter positive no45
end
>>>

2) else:-

Else block should be preceded by 'if' block or else-if block or while block or for block.

* If else block preceding block condition is returning false then only else block will be executed.

Syntax:-

if condition:-

Statement 1 SRI SAI BHAVANI XELLA
Statement 2 H.No. 7-1-209/1, Ground Floor,
 Ram Niya Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

Else :-

Statement 1
Statement 2

program:-

```
print ("begin")
x = input ("enter positive no")
i = int(x)
if i < 10:
    print ("given no is 1 digit no")
else:
    print ("given no is >= 2 digit no")
print ("end")
```

SRI SAI BHAVANI XELLA
H.No. 7-1-209/1, Ground Floor,
Ram Niya Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:-

```
begin
enter positive no25
given no. is >=2 digit no
end
>>>
begin
enter positive nos
given no is 1 digit no
end.
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

3) elif:-

elif should be proceeded by either if block or another elif block.

Syntax:-

if condition:

Statement 1

Statement 2

Elif condition:

Statement 1

Statement 2

* Elif block preceding block condition returns false then

control will comes to elif block

* After control is reaching elif if Elif block condition

returns true then only it will execute elif block.

Program:-

```
print("begin")
x = input("enter positive no")
i = int(x)
if i<10:
    print("given no is 1 digit no")
elif i<100:
```

```
print("given no is 2 digit no")
if i<100:
    print("given no is 3 digit no")
else:
    print("given no is >= 4 digit no")
print("end")
```

Op:-

```
begin
enter positive no45
given no is 2 digit no
end
>>
begin
enter positive no123
given no is 3 digit no
end
>>
```

Looping statements:-

Looping statements are used to execute set of statements repeatedly.

* python supports two looping statements. They are

- 1) while
- 2) For

1) While:-

While loop executes set of statements repeatedly

until condition becomes false:

Syntax:- while condition:

```
    Statement 1
    Statement 2
    ...
    ...
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Program:-

```
print("begin")
i=1
while i<=5:
    print("welcome")
    i=i+1
print("end")
```

O/p:-

```
begin
Welcome
Welcome
Welcome
Welcome
Welcome
end
>>>
```

program:-

```
print("begin")
i=1
sum=0
while i<=100:
    sum=sum+i
    i=i+1
print(sum)
print("end")
```

O/p:-

```
begin
5050
end
>>>
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Program:-

```
print ("begin")
i=1
while i<=5:
    print ("welcome")
    i=i+1
else:
    print ("in while else")
print ("end")
```

O/P:-

```
begin
welcome
welcome
welcome
welcome
welcome
welcome
in while else
end
>>>
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Break:-

Break is a statement, which can be used in looping statements. whenever control reach to the break statements of the loops then without executing the loop, control will comes out from the loop.

Ex:-

```
print ("begin")
i=1
while i<=5:
    print ("welcome")
    if i==3:
        break
    i=i+1
else:
    print ("in while else")
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

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

O/P:- begin
welcome
welcome
welcome
end
>>>

Break statement is used to exist from the infinite loops.

```
print("begin")
```

i=1

While True:

```
    print("welcome")
```

```
    if i == 3:
```

```
        break
```

```
    i = i + 1
```

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

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- begin

welcome

welcome

welcome

end

>>>

Continue:-

Continue is a statement, which can be used in looping statements

* Whenever control reaches to the continue statement of the looping statements then without executing the remaining part of that iteration, control will go to the next iteration.

```
i=0  
while i<5:  
    i = i+1  
    if i == 3:  
        continue  
    print ("welcome", i)
```

O/P:- welcome 1
welcome 2
welcome 4
welcome 5
>>>

Program:-

```
while True:  
    name = input ("enter user name")  
    if name != 'sathya':  
        continue  
    password = input ('Hello, sathya. What is the  
                      password?')  
    if password == 'lokesh':  
        break  
    print ('Access granted')
```

O/P:-
enter user name abc
enter user name xyz
enter user name sathya
Hello, Sathya. What is the password? abc
enter username sathya
Hello, sathya: what is the password? lokesh
Access granted

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Niira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

For loop:-

for loop executes set of statements with respect to every element of given iterable object.

Syntax:-

```
for var in IterableObject:
```

```
    - - - - -  
    - - - - -  
    - - - - -
```

Program:- $x = "sathya"$

```
for p in x:  
    print(p * 3)
```

O/p:-

```
sss  
aaa  
ttt  
hhh  
yyy  
aaa  
>>>
```

Example:- $a = \text{range}(10)$

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

$b = \text{range}(10, 20)$

```
for q in b:  
    print(q)
```

$c = \text{range}(20, 30, 2)$

```
for r in c:  
    print(r)
```

$d = \text{range}(40, 30, -1)$

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

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 010

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 010

O/P:-

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 22
- 24
- 26
- 28
- 40
- 39
- 38
- 37
- 36
- 35
- 34
- 33
- 32
- 31

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* Applying a common logic on objects is called as function
* Applying a particular logic on objects is called as

Method.

Working with methods of string:-

Example:- $x = \text{'sathya technologies'}$

`print(x)`

`print(len(x))`

`print(x.capitalize())`

`print(x.title())`

`print(x.upper())`

`print(x.lower())`

`print(x.isdigit())`

$y = \text{'1234'}$

`print(y)`

`print(y.isdigit())`

O/P:-

sathya technologies

19

Sathya technologies

Sathya Technologies

SATHYA TECHNOLOGIES

sathya technologies

False

1234

True

>>>

Collections:-

collection types represented classes objects are stored in group of objects (elements)

- * collection types represented classes objects are iterable objects.
- * Every collection type represented class provides methods to perform operations on elements of those objects.
- * python supports following collection types

- 1) list
- 2) tuple
- 3) set
- 4) dict

1) List:-

List objects can be created by using square brackets or by calling list function.

- * list objects are mutable objects
- * Inception order is preserved in list
- * Heterogeneous elements are allowed
- * Duplicate elements are allowed.
- * Every element in the list is represented with unique index
- * list supports both positive and negative indexing.

Example:-

```
a = []
print(a)
print(type(a))
print(len(a))

b = list()
print(b)
print(type(b))
print(len(b))
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

$c = [10, 20, 30, 40, 50]$ → Homogeneous.

print(c)

$d = [10, 12.12, \text{True}, 3+4j, \text{'sathya']}$ → Heterogeneous.

print(d)

$e = [1, 2, 3, 1, 2, 1]$ → duplicate (repeated no's)

print(e)

O/P:-

[]

<class 'list'>

0

[]

<class 'list'>

0

[10, 20, 30, 40, 50]

[10, 12.12, True, (3+4j), 'sathya']

[1, 2, 3, 1, 2, 1]

>>>

Example:-

$x = [10, 20, 30, 40, 50, 60]$

print(x)

print(x[2])

print(x[-2])

print(x[1:4])

print(x[4:1])

print(x[-4:-1])

$x[2] = 123$

print(x)

(0) 10 - 6
" 20 - 5
" 30 - 4
(1) 40 - 3
(2) 50 - 2
(3) 60 - 1

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,

Ram Mira Towers, Near Satyam Theatre,

Ameerpet, Hyderabad-500 016

O/P: [10, 20, 30, 40, 50, 60]

30

50

[20, 30, 40]

{ } ... insertion order should be given & print in the reverse
in backward order

[30, 40, 50]

[10, 20, 123, 40, 50, 60]

>>>

Example:

x = [10, 20, 30, 40, 50, 60]

print(x)

sum = 0

for p in x:

 sum = sum + p

print(sum)

i = 0

sum1 = 0

while i < len(x):

 sum1 = sum1 + x[i]

 i = i + 1

print(sum1)

O/P:

[10, 20, 30, 40, 50, 60]

210

210

>>>

SRI SAI BHAVANI XELLA
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

10+20+30
20+40
40+50
50+60

0+6
0+10
10+14
14+18
18+22

SRI SAI BHAVANI XELLA
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Nested List :-

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

for p in x:

 print(p, type(p))

 for q in p:

 print(q)

O/p:- [10, 20, 30] class <list>

 10

 20

 30

[40, 50, 60] class <list>

 40

 50

 60

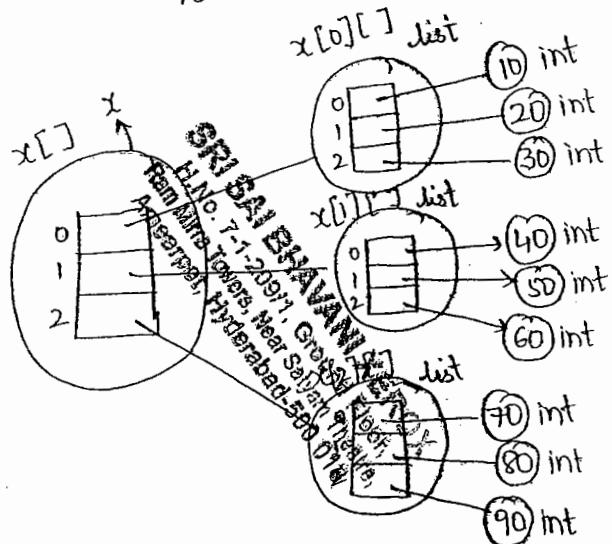
[70, 80, 90] class <list>

 70

 80

 90

SRI SAI BHAVANI XEROX
 H.No. 7-1-209/1, Ground Floor,
 Ram Mira Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016



Example:-

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

$i = \text{int}(\text{input}(\text{"Enter search element"}))$

$\text{if } i \text{ in } x:$

$\quad \text{print}(\text{"element is found"})$

else:

$\quad \text{print}(\text{"element is not found"})$

O/p:- Enter search element 40

element is found

>>>

Enter search element 60

element is not found

>>>

Unpacking Elements of list :-

Example:- $x = [100, \text{True}, 123, 123]$

$\text{print}(x, \text{type}(x))$

$a, b, c = x$

$\text{print}(a, \text{type}(a))$

$\text{print}(b, \text{type}(b))$

$\text{print}(c, \text{type}(c))$

O/p:- $[100, \text{True}, 123, 123] <\text{class 'list'}>$

100 <class 'int'>

True <class 'bool'>

123.123 <class 'float'>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Example:-

```

a='Sathya'
print(a)
x=list(a)
print(x)
x[2]='p'
print(x)
b=' '
for i in x:
    b=b+i
print(b)

```

Output:-

```

sathya
['s', 'a', 't', 'h', 'y', 'a']
['s', 'a', 'p', 'h', 'y', 'a']
Saphya
>>>

```

Working with methods of list:-

Example:- $x = [10, 20, 30, 40, 10]$

```

print(x)
x.append(50) → adding 50 to the list

```

```

print(x)
print(x.count(10))
print(x.index(30))

```

$y = x.copy()$

```

print(y)
x.insert(2, 123)

```

$print(x)$

$x.remove(30)$

$print(x)$

$x.pop(3)$ → remove 3rd element

SRI SAI BHAVANI XEL
 H.No. 7-1-209/1, Ground Floor,
 Ram Mitra Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEL
 H.No. 7-1-209/1, Ground Floor,
 Ram Mitra Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

```
print(x)
z = [80, 70, 90]
print(z)
x.extend(z)
print(x)
x.sort(reverse=True) desc
print(x)
x.reverse() asc
print(x)
x.clear() it is 0
print(x)
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- [10, 20, 30, 40, 10]
[10, 20, 30, 40, 10, 50]

2
1.
[10, 20, 30, 40, 10, 50]
[10, 20, 123, 30, 40, 10, 50]
[10, 20, 123, 40, 10, 50] *→ 123 is 40*
[10, 20, 123, 10, 50] *→ 123 is 40*
[80, 70, 90]
[10, 20, 123, 10, 50, 80, 70, 90] *it is 0*
[123, 90, 80, 70, 50, 20, 10, 10]
[10, 10, 20, 50, 70, 80, 90, 123]
[]
>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Range:- Range is pre defined function, is used to generate values for list comprehension.

List Comprehensions:-

The concept of generating the elements into the list object by writing some logic in the list is known as a list comprehensions.

Example:-

$x = [p \text{ for } p \text{ in range}(10)]$

print(x)

O/p:- $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

The above is equivalent to the following code

$x = []$

for p in range(10):

x.append(p)

print(x)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Example:-

$x = [p * p \text{ for } p \text{ in range}(10)]$

print(x)

$y = [q * q \text{ for } q \text{ in range}(10, 20) \text{ if } q \% 2 == 0]$

print(y)

O/p:- $[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]$

$[100, 144, 196, 256, 324]$.

Identifying primes

Example:-

no_primes = [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 no_primes]

print(primes)

(or)

no_primes = []

for i in range(2,8):

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

 no_primes.append(j)

primes = []

for x in range(2,50):

 if x not in no_primes:

 primes.append(x)

print(primes)

O/P:-

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

Working:-

no_primes = [4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50]

6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51

8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52

10, 15, 20, 25, 30, 35, 40, 45, 50

SRI SAI BHAVANI XE
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

12, 18, 24, 30, 36, 42, 48, 54

14, 21, 28, 35, 42, 49, 56

String Method of str class (split Method)

line = 'python@ is @ a@ opensource@ language'.

words = line.split('@')

for word in words:

 print(word)

O/p:- python

is

a

opensource

language

SRI SAI BHAVANIXE
 H.No. 7-1-209/1, Ground Floor,
 Ram Mohan Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 010

Example:-

line = 'The quick brown fox jumps over the lazy dog'

words = line.split()

stuff = [[w.upper(), w.lower(), len(w)] for w in words]

for i in stuff:

 print(i).

O/p:- ['THE', 'the', 3]

['QUICK', 'quick', 5]

['BROWN', 'brown', 5]

['FOX', 'fox', 3]

['JUMPS', 'jumps', 5]

['OVER', 'over', 4]

['THE', 'the', 3]

['LAZY', 'lazy', 4]

['DOG', 'dog', 3]

SRI SAI BHAVANIXE
 H.No. 7-1-209/1, Ground Floor,
 Ram Mohan Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 010

TUPLE:-

Tuple objects can be created by using parenthesis or by calling tuple function or by assigning multiple values to a single variable.

* tuple objects are ^{> doesn't modify} immutable objects

* Incession order is preserved

* Duplicate elements are allowed

* Heterogeneous elements are allowed

* tuple supports both positive and negative

* The elements of the tuple

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Example:-

x=()

print(x)

print(type(x))

print(len(x))

y=tuple()

print(y)

print(type(y))

print(len(y))

z=10,20

print(z)

print(type(z))

print(len(z))

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

print(p)

q=(100,123,123, True, "sathya")

print(q)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- ()

<class 'tuple'>

0

()

<class 'tuple'>

0

(10, 20)

<class 'tuple'>

2

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

(100, 123, 123, True, 'Sabya')

>>>

Example:-

x = [10, 20, 30, 40, 50, 60]

print(x)

print(x[2])

print(x[-2])

print(x[2:5])

print(x[-6:-2])

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:-

(10, 20, 30, 40, 50, 60)

30

50

(30, 40, 50)

(10, 20, 30, 40)

Ex:-

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

sum=0

for p in x:

 sum = sum + p

print(sum)

sum1=0

i=0

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

while $i < \text{len}(x)$:

sum1 = sum1 + x[i]

$i = i + 1$

print (sum1)

O/P:- 150
150
>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Niya Towers, Near Satyam Theatre,
Amarapet, Hyderabad-500 016

Example:-

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

print(x)

for p in x:

print(p)

for q in p:

print(q)

O/P:- ((10, 20, 30), (40, 50, 60), (70, 80, 90))

(10, 20, 30)

10

20

30

(40, 50, 60)

40

50

60

(70, 80, 90)

70

80

90

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Niya Towers, Near Satyam Theatre,
Amarapet, Hyderabad-500 016

List in tuple:-

Example:-

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

print(x)

for p in x:

 print(p, type(p))

$$x[1][1] = 100$$

print(x)

O/P:- ((10, 20, 30), [40, 50, 60], (70, 80, 90))

(10, 20, 30) <class 'tuple'>

[40, 50, 60] <class 'list'>

(70, 80, 90) <class 'tuple'>

((10, 20, 30), [40, 100, 60], (70, 80, 90))

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Example:-

$$x = (10, 12.12, \text{True})$$

print(x)

a, b, c = x

print(a, type(a))

print(b, type(b))

print(c, type(c))

$$x[1] = 1000$$

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- (10, 12.12, True)

10 <class 'int'>

12.12 <class 'float'>

True <class 'bool'>

Error 'tuple' is immutable object, it does not
support item assignment.

Example:-

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

Print (x)

Print ($x.$ index (30))

Print ($x.$ index (10, 2)) → it goes to the second element of containing

Print ($x.$ count (10)) the value of 10

Print ($x.$ count (40))

O/P:- (10, 20, 30, 40, 10, 60, 20)

2

4

2

1

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Differences b/w List and Tuple

List

- 1) List objects are mutable objects
- 2) Applying iterations on list objects takes longer time
- 3) If the frequent operation is insertion or deletion of the elements then it is recommended to use list
- 4) List can't be used as a 'key' for the dictionary.

Tuple

- 1) tuple objects are immutable objects.
- 2) Applying iterations on tuple objects takes less time.
- 3) If the frequent operation is retrieval of the elements then it is recommended to use tuple
- 4) tuple can be used as a key for the dictionary if the tuple is storing only immutable elements.

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SET:-

Set objects can be created by using curly braces {} or by calling set function.

- * Insertion order is not preserved in set.
- * Duplicate elements is not allowed.
- * Heterogeneous elements are allowed in set.
- * Set is not supporting indexing.
- * Set objects are mutable objects.
- * Elements of the set must be immutable.
- * We can perform the mathematical set operations like union, intersection and difference and symmetric difference on set objects.

Ex:- $x = \{10\}$

```
print(x)
print(type(x))
print(len(x))
```

$y = \text{set}()$

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

$\text{print}(\text{len}(y))$

$z = \{10, 20, 30, 40, 50\}$

$\text{print}(z)$

$p = \{10, 20, 10, 20, 10, 30\}$

$\text{print}(p)$

$q = \{100, 12.12, \text{True}, "sathya", 3+5j\}$

$\text{print}(q)$

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

WEEKEND
WEEKEND
WEEKEND

Q1P:- {10}

<class 'set'>

1

set()

<class 'set'>

0

{40, 10, 20, 50, 30}

{10, 20, 30}

{12.12, (3+5j), 100, True, 'sathya'}

>>>

* tuple can be stored in set, as it is immutable object.

* Set in set also not possible in set, because set in set is immutable

Ex:- x = {10, 20, 30, 40, 50}

print(x)

print(30 in x) → True

for p in x:

 print(p)

y = {(10, 20, 30), (40, 50, 60), (70, 80, 90)}

print(y)

for q in y:

 print(q, type(q))

Q1P:- {40, 10, 20, 50, 30}

True

40

10

20

50

30

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

$\{(10, 20, 30), (40, 50, 60), (70, 80, 90)\}$

(10, 20, 30) <class 'tuple'>

(40, 50, 60) <class 'tuple'>

(70, 80, 90) <class 'tuple'>

>>>

* If tuple contains immutable elements then only we can use the tuple in the set

Ex:- $y = \{(10, 20, 30), (40, 50, 60, [1, 2, 3]), (70, 80, 90)\}$
print(y)

O/P:- Error <'mutable elements are not supported i.e., 'list''>

* Discard method will not display any error if given no is not 'set'.

* Remove method will display 'key error' if given no is not in set.

Ex:- $x = \{10, 20, 30, 40, 50\}$

print(x)

x.add(60)

print(x)

y = x.copy()

print(y)

x.remove(20)

print(x)

x.pop() \rightarrow item no. will be removed

x.pop()

print(x)

x.discard(30)

print(x)

x.clear()

print(x)

O/P:-

- { 40, 10, 20, 50, 30 }
- { 40, 10, 50, 20, 60, 30 }
- { 50, 20, 40, 10, 60, 30 }
- { 40, 10, 50, 60, 30 }
- { 10, 50, 60, 30 }
- { 10, 50, 60 }

Set()

>>>

Example:- A = { 1, 2, 3, 4, 5 }

print(A)

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

print(B)

print(A|B)

print(B|A)

print(A.union(B))

print(B.union(A))

print(A & B)

print(B & A)

print(A.intersection(B))

print(B.intersection(A))

print(A - B)

print(B - A)

print(A.difference(B))

print(B.difference(A))

print(A ^ B)

print(B ^ A)

print(A.symmetric_difference(B))

print(B.symmetric_difference(A))

SRI SAI BHAVANI XEP
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEP
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Op:-

$\{1, 2, 3, 4, 5\}$
 $\{8, 4, 5, 6, 7\}$
 $\{1, 2, 3, 4, 5, 6, 7, 8\}$
 $\{1, 2, 3, 4, 5, 6, 7, 8\}$
 $\{1, 2, 3, 4, 5, 6, 7, 8\}$
 $\{4, 5\}$
 $\{4, 5\}$
 $\{4, 5\}$
 $\{4, 5\}$
 $\{1, 2, 3\}$
 $\{8, 6, 7\}$
 $\{1, 2, 3\}$
 $\{8, 6, 7\}$
 $\{1, 2, 3, 6, 7, 8\}$
 $\{1, 2, 3, 6, 7, 8\}$
 $\{1, 2, 3, 6, 7, 8\}$
 $\{1, 2, 3, 6, 7, 8\}$

>>>

Example:-

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

print(x)

y = set(x)

print(y)

p = 'sathya'

print(p)

q = set(p)

print(q)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Rani Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/p:- [10, 20, 10, 30, 10, 20] → Jdt

{10, 20, 30} → sets do not allow duplicate values

Sathya

{'y', 'h', 't', 's', 'a'}

>>>

Set Comprehension:-

SRI SAI BHAVAN

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * The concept of generating the elements into the set object by writing some logic in the set is known as 'set comprehension'

Example:- $x = \{p \text{ for } p \text{ in range}(10)\}$

print(x)

$y = \{q * q \text{ for } q \text{ in range}(10, 20) \text{ if } q \% 2 == 0\}$

print(y)

$z = \{x \text{ for } x \text{ in range}(20, 30) \text{ if } x \% 2 != 0\}$

print(z)

O/p:- {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

{144, 256, 196, 100, 324}

{25, 29, 27, 21, 23}

Dict:-

SRI SAI BHAVANI XECUTIVE

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * dictionary objects can be created by using curly braces {}

or by calling dict function

* dictionary objects are mutable objects

* dictionary represents key-value base

* Each key-value pair of dictionary is known as a item.

* Dictionary keys must be immutable

* Dictionary values can be mutable or immutable.

- * Duplicate keys are not allowed but values can be duplicate
- * Insertion order is not preserved.
- * Heterogeneous keys and Heterogeneous values are allowed.

Example:- $x = \{ \}$

print(x)

print(type(x))

print(len(x))

y = dict

print(y)

print(type(y))

print(len(y))

$z = \{ \text{"python": 99, "java": 90, "oracle": 82, "django": 97} \}$

print(z)

O/P:- $\{ \}$

<class 'dict'>

0

$\{ \}$

<class 'dict'>

0

$\{ \text{'oracle': 82, 'python': 99, 'java': 90, 'django': 97} \}$

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Example:-

$x = \{ "python": 99, "java": 90, "oracle": 82, "django": 99 \}$

`print(x)`

$x["oracle"] = 92$

`print(x)`

$y = \{ 100: "hyd", 12.12: "bang", \text{True}: "pune", 3+4j: "chennai" \}$

`print(y)`

$z = \{ 10: 1000, 20: 12.12, 30: \text{True}, 40: 3+5j \}$

`print(z)`

$p = \{ (1,2,3): "flask", "hyd": [5,6,7] \}$

`print(p)`

O/P:-

$\{ "oracle": 82, "java": 90, "python": 99, "django": 99 \}$

$\{ "oracle": 92, "java": 90, "python": 99, "django": 99 \}$

$\{ 12.12: "bang", (3+4j): "chennai", 100: "hyd", \text{True}: "pune" \}$

$\{ 40: (3+5j), 10: 1000, 20: 12.12, 30: \text{True} \}$

$\{ "hyd": [5,6,7], (1,2,3): "flask" \}$

`>>>`

Ex:- $x = \{ "python": 99, "java": 90, "oracle": 82, "django": 99 \}$

`print(x)`

$x["pyramid"] = 87$

`print(x)`

`print(x["python"])`

`print(x.get("django"))` → by calling method

`x.pop("oracle")` → to remove

`print(x)`

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
x.popitem()
print(x)
K = x.keys()
```

```
print(K)
```

```
V = x.values()
```

```
print(V)
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- {'java':90, 'django':99, 'python':99, 'oracle':82}

{'java':90, 'pyramid':87, 'django':99, 'python':99, 'oracle':82}

99

99

{'java':90, 'pyramid':87, 'django':99, 'python':99}

{'pyramid':87, 'django':99, 'python':99}

dict_keys(['pyramid', 'django', 'python'])

dict_values([87, 99, 99])

>>>

Items Method:-

```
x = {"python":99, "java":90, "oracle":82, "django":99}
```

```
print(x)
```

```
KV = x.items()
```

```
print(KV)
```

```
sum = 0
```

```
for p in KV:
```

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

```
    sum = sum + p[1]
```

```
print("total", sum)
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- { 'oracle': 82, 'python': 99, 'django': 99, 'java': 90 }

dict-items([('oracle', 82), ('python', 99), ('django', 99), ('java', 90)])

oracle 82

python 99

django 99

java 90

total 370

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Niya Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Dictionary Comprehensions:-

- * The concept of generating items into the dictionary by writing some logic in the dictionary is known as a 'dictionary comprehension'

Ex:- $x = \{ p : p * p \text{ in range}(5) \}$

print(x)

$y = \{ q : q * q \text{ in range}(5, 15) \text{ if } q \% 2 == 0 \}$

print(y)

O/P:- { 0:0, 1:1, 2:4, 3:9, 4:16 }

{ 9:81, 11:121, 13:169, 5:25, 7:49 }

>>>

Functions:-

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

* Function will not be executed automatically.

* Function will be executed whenever we make a function call.

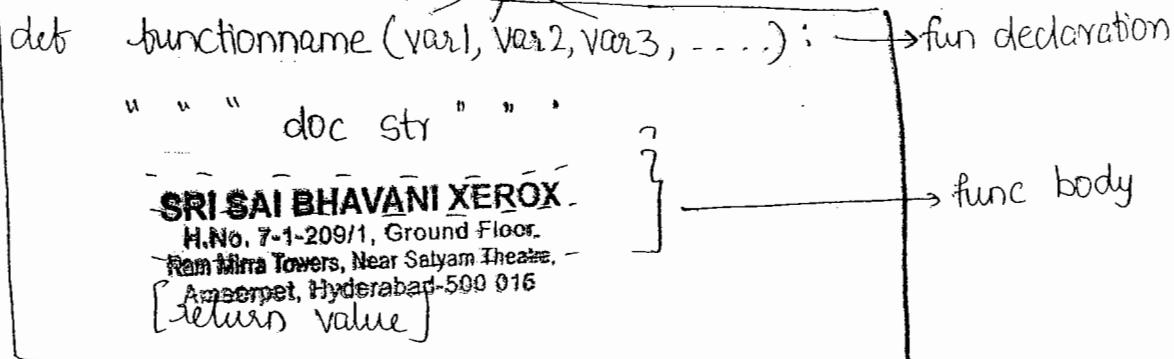
* One function we call for 'n' number of times.

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Niya Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Syntax :-

parameters



Example :- print ("begin")

def f1 (): *func declaration*

print ("in f1")

f1 () *func calling*

f1 ()

print ("end")

pre-defined values - return values

user-defined values - print values

O/P:- begin
in f1
in f1
end
>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Documentation string :-

* documentation string is used to provide the description about the function.

* Within the function documentation string is optional but it is recommended to define documentation string in the function.

Ex:- def f1 ()

"""
 sample function to test doc string
 """

print ("in f1")

help (len)

help (f1)

Parameters:-

- * The variables which are declared within the function declaration are known as parameters.
- * At the time of calling the function, we have to pass the values to the parameters of the function.
- * Parameters of a function can be accessed within the same function only.
- * Within the function declaration, parameters are optional
- * Python supports two types of parameters. They are
 - 1) Non default parameters
 - 2) default parameters.

Non default parameters:-

- * The parameters which are declared without assigning any value are known as non default parameters.
- * At the time of calling the function we should pass values of the non default parameters of the function.

Ex:- def add(a, b):

$$c = a + b$$

print(c)

add(10, 20)

add(100, 200)

add(123, 456)

add()

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Raja Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:-
30
300
519
Error
>>>

Default parameters:-

- * The parameters which are declared by assigning some value are known as default parameters.
- * At the time of calling the function we need not to pass the default values to the parameters of a function.

Ex:- def add(a=10, b=20):

$$c = a + b$$

print(c)

add()

add(1000)

add(100, 200)

add(123, 456)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:-

30

1020

300

579

>>>

Ex:- def add(^{nondet}a, ^{default}b=20):

$$c = a + b$$

print(c)

add(1000)

add(100, 200)

add(123, 456)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- 1020

300

579

>>>

* After defining the default parameters we are not allowed to define non default parameters.

Ex:- def add (a=10, b):

c = a+b

print(c)

add (1000)

add (100, 200)

add (123, 456)

SRISAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- syntax error

Arguments:-

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

* Python supports two types of arguments. They are

- 1) Non keyword arguments.
- 2) Keyword arguments.

1) Non keyword arguments:-

The arguments which are passed without assigning to the parameters are known as 'nonkey word argument'

Ex:- def greet (name, msg):

 print ("hello", name, msg)

greet ("sathya", "good morning")

greet ("good evening", "lokesha")

SRISAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- hello sathya good morning

hello good evening lokesha

>>>

Key word Argument:-

The arguments which are passed by assigning to parameter name are known as Keyword Arguments.

Ex:- def greet(name, msg):

```
    print("Hello", name, msg)
```

```
greet(name = "Sathya", msg = "good morning")
```

```
greet(msg = "good evening", name = "vasavi")
```

O/P:- Hello sathya good morning

Hello vasavi good evening

* At the defining keyword arguments not allowed to define non keyword arguments

Ex:- def greet(name, msg):

```
    print("Hello", name, msg)
```

```
greet("sathya", msg = "good morning")
```

```
greet(msg = "good evening", "laksh")
```

O/P:- Syntax error

Orbitary parameters:-

The parameters which are preceded by star(*)

are known as orbitary parameters.

* Orbitary parameter type internally taken as tuple.

* At the time of calling of the function we can pass zero or more values to the orbitary parameters.

Ex:- def add (*a):

 print(a)

 print(type(a))

 Sum=0

 for p in a:

 Sum = Sum+p

 print (sum)

add ()

add (100)

add (10, 20)

add (1, 2, 3, 4, 5)

O/P:- ()

<class 'tuple'>

0

(100,)

<class 'tuple'>

100

(10, 20)

<class 'tuple'>

30

(1, 2, 3, 4, 5)

<class 'tuple'>

15

>>>

Ex:- def add (a, *b):

 print(a)

 print(b)

add (100)

add (10, 20)

add (1, 2, 3, 4, 5)

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:-

1000
()
10
(20,)
1
(2,3,4,5)
>>>

Ex:- def add(*a, b):

 print(a)
 print(b)
 add (b= 1000)
 add (10, b= 20)
 add (1,2,3,4, b= 5)

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:-
()
1000
(10,)
20
(1,2,3,4)
5
>>>

Return Statement:-

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * We can define 'n' no of return statements in a function
- * Whenever control reach to the return statement of a function then without executing remaining part of the function control will comes out from function execution by returning some value.
- * The return value of the function can be stored into a variable & we can use that variable in the remaining part of the function.
- * If we are not storing the return value of a function in any variable then that data will become as a garbage collector.

Ex:- def add(a, b):

$$C = a + b$$

return C

$$x = \text{add}(100, 200)$$

print(x)

$$\text{add}(10, 20)$$

O/P:-

$$300$$

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- def absolute(a):

if a < 0:

return -a

else

return a

print("end of absolute")

$$x = \text{absolute}(-5)$$

print(x)

$$y = \text{absolute}(15)$$

print(y)

O/P:-

$$5$$

$$15$$

>>>

* By default function returns the "none" value

Ex:- def add(a, b):

$$C = a + b$$

$$x = \text{add}(10, 20)$$

print(x)

O/P:-

None

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * For every function internally one object is created, store that object address into the function name represented variable and we can call that function, through that variable.
- * Function object address which is present inside function name variable we can copy into other variables & we can call that function through that variable

Ex:-

```

def myfunc():
    print("welcome")
print(myfunc)
myfunc()
x = myfunc
print(x)
x()

```

SRI SAI BHAVANI XEROX
H.No. 7-1-202/1, Ground Floor,
Ram Mira Towers, Near Satyan Theatre,
Ameerpet, Hyderabad-500 016

Op:-

```

<function myfunc at 0x023D97C8>
welcome
<function myfunc at 0x023D97C8>
welcome

```

Global Variables:-

- The variable which are declared outside of all the functions are known as global variables.
- * Global variables of one program can be accessed within all the functions of the same program.
 - * Global variables of a program memory will be allocated only once.

SRI SAI BHAVANI XEROX
H.No. 7-1-202/1, Ground Floor,
Ram Mira Towers, Near Satyan Theatre,
Ameerpet, Hyderabad-500 016

Local Variables:-

- * The variables which are declared within the function are known as local variables.
- * Local variables of one function can't be accessed outside of that function.
- * For all the local variables of a function memory will be allocated with respect to every time function.

Ex:- a=1000 → G.V

def f1():

b = 2000 → L.V

print(a)

print(b)

def f2():

c = 3000 → L.V

print(a)

print(c)

f1()

f2()

O/P:- 1000

2000

1000

3000

>>>

- * Inorder to modify the global variable values within the function we should define the forward declaration of global variables.

SRI SAI BHAVANIKERIKA

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 018

SRI SAI BHAVANIKERIKA

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 018

Ex:-

```
a=1000
def f1():
    global a
    a=2000
    print(a)
def f2():
    print(a)
```

f1()

f2()

O/P:-

2000

2000

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Nirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Recursive Function invocation:-

- * A function which is called by itself is known as a recursive function invocation
- * Infinite recursion is not supported by python.

Ex:- def f1():

```
    print("in f1")
```

```
    f1()
```

```
f1()
```

O/P:- in f1

in f1

in f1

!

!

!

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Nirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- i=0

```
def f1():
```

```
    print("in f1")
```

```
    global i
```

```
    i = i + 1
```

```
    while i <= 5
```

```
        f1()
```

f1()

OP:- In f1

In f1

In f1

In f1

In f1

In f1

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Anonymous function or lambda function:-

- * A function which doesn't contain any name is known as a Anonymous function or lambda function.

Syntax:-

lambda arguments : expression

- * Lambda function we can assign to the variable & we can call the lambda function through the variable.

Ex:- myfunc = lambda x: x*x

a = myfunc(10)

print(a)

OP:- 100

>>>

- * The above function is equivalent to the following function

Ex:- def myfunc(x):
 return x*x

a = myfunc(10)

print(a)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

OP:- 100

>>>

Ex:- def cube(a, square):

 sq = square(a)

 return a * sq

x = cube(10, lambda b: b*b)

print(x)

O/P:- 1000

>>>

Ex:- my_list = [1, 5, 4, 6, 8, 11, 13, 12]

new_list = list(filter(lambda x: (x % 2 == 0), my_list))

print(new_list)

O/P:- [4, 6, 8, 12]

>>>

Ex:- my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(map(lambda x: (x * x), my_list))

print(new_list)

O/P:- [1, 25, 16, 36, 64, 121, 9, 144]

>>>

* filter gives the values, based on the condition whether it is true or false. It displays only true values.

* map gives the values whether it is true or false.

It displays both true and false.

Modules:-

* Every python file itself is known as a module

* A module can contain global variables, executable

statement, functions, and classes.

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * The properties of one module we can use into another module by using 'import' statement.
- * python supports two types of import statements
 - 1) Normal import
 - 2) From import

I) Normal import:- In normal (module) import entire python file or module object is imported.

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

mod1.py

```
a=1000
def f1():
    print("in f1 of mod1")
```

SRI SAI SHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

mod2.py

```
import mod1
import math
b=2000
def f2():
    print("in f2 of mod2")
    print(b)
f2()
print(mod1.a)
mod1.f1()
print(math.pi)
x=math.sqrt(100)
print(x)
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/p:-

2000

in f2 of mod2

1000

in f1 of mod1

3.141592653589793

10.0

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * At the time of importing a module by using normal import statement we can create the alias name for the module and we can access the properties of that module through alias name.
- * After creating the alias name for a module we are not allowed to access the properties of that modules through its original name.

mod2.py

```
import mod1 as p
```

```
import math as q
```

```
b = 2000
```

```
def f2():
```

```
    print("in f2 of mod2")
```

```
print(b)
```

```
f2()
```

```
print(p.a)
```

```
p.f1()
```

```
print(q.pi)
```

```
x = q.sqrt(100)
```

```
print(x)
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/p:- 2000

in f2 or mod2

1000

in f1 or mod1

3.141592653589793

10.0

>>>

From import:-

* By using from import we can import the required properties of a module

* The properties which are imported by using 'from' import we can access those properties directly without using module name or alias name.

Ex:- from mod1 import a, f1

from math import pi, sqrt

b=2000

def f2 ():

 print("in f2 or mod2")

print(b)

f2()

print(a)

f1()

print(pi)

x = sqrt(100)

print(x)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/p:- 2000

in f2 or mod2

1000

in f1 or mod1

3.141592653589793

10.0

>>>

```

Ex:- from mod1 import *
from math import *
b=2000
def f2():
    print("in f2 of mod2")
print(b)
f2()
print(a)
f1()
print(pi)
x = sqrt(100)
print(x)

```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/p:-

```

2000
in f2 of mod2
1000
in f1 of mod1
3.14159265
10.0
>>>

```

Built-In's Module:-

- * Built-in's is a predefined module, which contains frequently used functions. python interpreter imports built-in's module for each and every python file automatically.
- * We can use the properties of built-in-modules directly without importing built-in-modules explicitly

Example:-

```

import math
print(math.pi)
print(math.sqrt(100))
print(len("sathya"))

```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

op:- 3.14159265

10.0

6

>>>

Re-loading a module:-

- * Even though we import one module for more than once then by default that module is imported for only once.
- * To load the already loaded module into the memory location we use reload function of imp module
- * imp module is a predefined module

mod3.py

```
print("hi from mod3")
```

```
a=1000
```

```
def f1():
```

```
    print("in f1 of mod3")
```

mod4.py

```
import mod3
```

```
import imp
```

```
b=2000
```

```
def f2():
```

```
    print("in f2 of mod4")
```

```
print(b)
```

```
f2()
```

```
print(mod3.a)
```

```
mod3.f1()
```

```
imp.reload(mod3)
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Q1:- hi from mod3
2000
in f2 of mod4
1000
in f1 of mod3
hi from mod3
>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Modules search path:-

- * By default python interpreter search for the imported modules in the following locations:-
 - 1) Current directory (main module location)
 - 2) Environment variable path
 - 3) Installation dependent directory.
- * If the imported module is not found in the any one of the above locations. Then python interpreter gives error

Built-in attributes of a module:-

- * By default for each and every python module some properties are added internally and we call those properties as a built-in-attributes of a module

Ex:- import mod3
from math import pi, sqrt
b=2000
def f2():
 print("in f2 of mod4")
print(dir())

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

OP:- hi from mod3

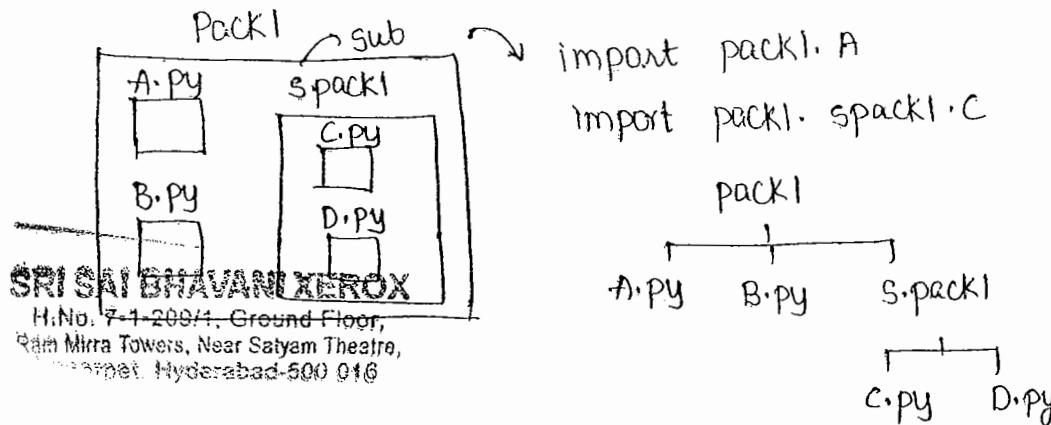
```
[('__annotations__', '__builtins__', '__doc__', '__file__',
  '__loader__', '__name__', '__package__', '__spec__', 'b',
  'f2', 'mod3', 'pi', 'sqrt'])
```

Packages:-

- * Package is nothing but a bolder or directory which represents collection of modules.
- * A package can also contain sub packages.
- * We can import the modules of the package by using `packagename.module name`
- * We can import the modules of the subpackages by using `packagename.subpackage name.module name`

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016



Structure of package:-

A.py

a=1000

```
def f1():
    print("in f1 of A of pack1")
```

B.py

b=2000

```
def f2():
    print("in f2 of B of pack1")
```

C.py

c = 3000

def f3():

print(" in f3 of c of pack1.spck1")

D.py

d = 4000

def f4():

print(" in f4 of D of pack1.spck1")

Test.py

import pack1.A

from pack1.B import b, f2

import pack1.spck1.C as P

from pack1.spck1.D import *

print(pack1.A.a)

pack1.A.f1()

print(b)

f2()

print(P.c)

P.f3()

print(d)

f4()

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/p:- 1000

in f1 of A of pack1

2000

in f2 of B of pack1

3000

in f3 of C of pack1.spck1

4000

in f4 of D of pack1.spck1

>>>

File Handling:-

- * File is a named location on the disk, which stores the data in permanent manner
- * Python language provides various functions and methods to provide the communication between Python programs and files.
- * Python programs can open the file, perform the read or write operations on the file and close the file
- * We can open the files by calling open function of built-in modules
- * At the time of opening the file, we have to specify the mode of the file
- * Mode of the file indicates for what purpose the file is going to be opened (r, w, a, b)

File modes

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

| <u>Mode</u> | <u>Description</u> |
|-------------|--|
| 'r' | - open a file for reading (default) |
| 'w' | - open a file for writing. Creates a new file if it does not exist or truncates the file if it exists |
| 'x' | - open a file for exclusive creation. If the file already exists the operation fails |
| 'a' | - open for appending at the end of the file without truncating it. Creates a new file if it does not exist |
| 't' | - open in text mode (default) |
| 'b' | - open in binary mode |
| '+' | - open a file for updating (reading & writing) |

- * 'open function' open the file in the specified mode and creates file object.
- * File object provides various methods and by using those methods we can perform the read or write operations on the files and we can close the files.

Eg:- `x = open("myfile.txt")`
`print(x.read())`
`x.close()`

O/P:- sathyatech
ameerpet
hyderabad
telangana
>>>

Eg:- `x = open("myfile.txt")`
`print(x.tell())` → to print cursor position
`x.seek(8)` → it indicates the file cursor to read from 8th position
`print(x.tell())`
`print(x.read())`
`print(x.tell())`
`x.close()`

O/P:-
0
8
ech
ameerpet
hyderabad
telangana
43
>>>

Eg:- `x=open("myfile.txt")`
`print(x.tell())`

`x.seek(8)`
`print(x.tell())`

`print(x.readline())` → it will read the data from current line to end of the line.

`print(x.tell())`
`x.close()`

O/P:-
0
8
ech

13
>>>

Eg:- `x=open("myfile.txt")`

`lines=x.readlines()` → to read in single string line format

for p in lines:

`print(p)`

`x.close()`.

O/P:- sathya tech

ameerpet

hyderabad

telangana

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Waiting the data into the file

Ex:- `x=open("myfile.txt","w")
x.write("sathya tech In ameerpet \n")
x.close()`

O/P:- Copy will be stored in python file (myfile)

Ex:- `x=open("myfile.txt","a")
x.write("hyderabad In telangana")
x.close()`

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- Copy will be stored in pythonfile (myfile)

Exception Handling:-

* Generally we get the two types of errors in programming languages. They are

- 1) Syntax errors
- 2) Runtime errors.

1) Syntax errors:-

* The errors which occurs because of invalid syntaxes is known as syntax errors.

* Whenever syntax error is occurred in python program, then compiled python file will not be generated

* Without getting compiled python file program execution will not be started

* Programmer is responsible for providing the solutions to the syntax error

Ex:- `def f1()
print(" in f1")`

O/P:- Syntax error:-

After function declaration, we should mention block

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Run time errors:-

- * The errors which occurs after starting the execution of the program are known as runtime errors.
- * Runtime errors can occur because of
 - i) Invalid I/p
 - ii) Invalid logic
 - iii) memory issues
 - iv) Hardware failures and so on.
- * With respect to every reason which causes to runtime error corresponding runtime error representation class is available
- * Runtime error representation classes technically we call as a exception classes.
- * While executing the program if any runtime error is occur corresponding run time error representation class object is created
- * Creating run time error representation class object is technically known as a rising exception
- * While executing the program if any exception is rised, then internally python interpreter verifies any code is implemented to handle rised exception or not
- * If code is not implemented to handle rised exception then program will be terminated abnormally.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Normal termination:-

- * The concept of terminating the program in the middle of its execution without executing last statement of the main module is known as abnormal termination.
- * Abnormal termination is undesirable situation in programming languages.

Ex:- print("begin")

i = input("enter fno")

j = input("enter sno")

x = int(i)

y = int(j)

z = x/y

print(z)

print("end")

O/P:- begin

enter fno 100

enter sno 20

z = 5.0

end

(or)

O/P:- begin

enter fno 100

enter sno 0

zero division error: division by zero.

Exception Handling:-

- * The concept of identifying raised exception, receiving that exception and assigning that exception to corresponding exceptional class is known as exception handling.
- * Exception handling can be implemented by using 'try' and 'except' blocks.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Try block:-

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

Syntax:- try:

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * The statements which causes to run time errors and other statements which depends on the execution of run time errors statements are recommended to represent in try block
- * While executing try block statement if any exception is rised then immediately try block identifies that exception, receive that exception and forward that exception to except block without executing remaining statements to try block.

Except Block:-

- * A block which is preceded by the except keyword is known as except block.
- * Except block should be preceded by try block

Except (Exception class):

- * Except block requires the exception object given by the try block and assign that exception to corresponding exception class.
- * In except block we can also define the statements to display the user friendly error messages
- * If exception is rised in try block statements then only control comes to the except block.

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:-

```
print ("begin")
i = input ("enter fno")
j = input ("enter sno")
x = int(i)
y = int(j)
try :
    z = x/y
    print(z)
```

except (ZeroDivisionError): → This is named except block

```
    print ("sno can not be zero")
    print ("end")
```

O/P:-

```
begin
enter fno 100
enter sno 20
5.0
end
>>>
```

O/P:-

```
begin
enter fno 100
enter sno 0
sno can not be zero
end
>>
```

O/P:-

```
begin
enter fno abc
enter sno 100
```

Value Error

Default Except Block:-

Default except block is a except block which can handle any type of exception

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```

Ex:- print("begin")
      i = input("enter fno")
      j = input("enter sno")
      try:
          x = int(i)
          y = int(j)
          z = x * y
          print(z)
      except:
          print("error occurred")
      print("end")
  
```

O/p:-

- begin
- enter fno 100
- enter sno 0
- error occurred
- end
- >>>

(or)

O/p:-

- begin
- enter fno 100
- enter sno abc
- error occurred
- end
- >>>

Single try with multiple except blocks:-

- * To handle different exceptions by using different except block we use single try in multiple except blocks.
- * Whenever we define single try in multiple except blocks if any exception is raised in try block then control goes to the first except block.
- * If the first except block is not handle the exception then only control will goes to the next except block.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- print ("begin")

i = input ("enter fno")

j = input ("enter sno")

try :

x = int(i)

y = int(j)

z = x/y

print(z)

except (ZeroDivisionError) :

print ("sno can not be zero")

except (ValueError) :

print ("enter numerical values only")

except :

print ("error occurred")

print ("end")

O/P:- begin

enter fno 100

enter sno 0

sno can not be zero

end

>>>

O/P:- begin

enter fno 100

enter sno abc

enter numerical values only

end

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* While defining single try with multiple except blocks a common except block should be the last except block otherwise we get syntax error.

Ex:- print("begin")

i = input("enter sno")

j = input("enter sno")

try:

x = int(i)

y = int(j)

z = x/y

print(z)

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

except:

print("error occurred")

except(zeroDivisionError):

print("sno can not be zero")

except(ValueError):

print("enter numerical values only")

print("end")

O/p:- Syntax error

Finally block:-

* A block which is preceded by the finally keyword is known as a finally block

Syntax:-

finally:

- - -
- - --
- - --

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* Finally block should be preceded by either try block or except block.

* The set of statements which are compulsory to execute whether exception is raised or not raised even though exception

exception is raised or not raised even though exception

is raised whether it is handled or not handled are recommended to represent in finally block.

* Resource releasing statements (file closing statements) or (database connection close statements) are recommended to represent in finally block.

Ex:- print("begin")

i = input("enter fno")

j = input("enter sno")

try:

x = int(i)

y = int(j)

z = x/y

print(z)

except (ZeroDivisionError):

print("sno can not be zero")

finally:

print("welcome")

print("end")

O/P:-

begin

enter fno 100

enter sno abc

Welcome

Error

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 018

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 018

Ex:- `x = None`

`try:`

`x = open("myfile.txt")`

`print("file is opened")`

`print(x.read())`

`except:`

`print("error occurred")`

`finally:`

`if x != None`

`x.close()`

`print("file is closed")`

Op:- file is opened

Sathya tech

ameerpet

hyderabad

telangana

file is closed.

Nested try blocks:-

- * The concept of defining one try block inside another try block is known as nested try block.
- * A try block which contains another try block is known as a outer try block.
- * A try block which contains is present in another try block is known as inner try block.
- * If the exception is raised in outer try block then control will go to the outer try except block.
- * If outer try except block is not handle that exception then program will be terminated abnormally.
- * If the exception is raised in inner try block then control goes in inner try except block.
- * If the inner try except block is not handle that exception then control will goes to outer try except block.

SRISAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- try:

 print(" in try 1")

try:

 print(" in try2")

except:

 print(" in except 2")

finally:

 print(" in finally2")

except:

 print(" in except1")

try:

 print(" in try 3")

except:

 print(" in except 3")

finally:

 print(" in finally 3")

finally:

 print(" in finally1")

try:

 print(" in try 4")

except:

 print(" in except 4")

finally:

 print(" in finally4")

Op:-

in try1

in try2

in finally2

in finally1

in try4

in finally

~ ~ ~

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Amarpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Amarpet, Hyderabad-500 016

User defined Exception:-

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

Steps to implement User defined Exception

1) Defining User defined Exception class :-

* Defining any user defined class by extending any one of the pre defined exception class is known as user defined exception class

2) Raising Exception :-

* Creating exception class object by using raise keyword is known as a raised exception

3) Handling the raised exception :-

* we can handle the raised exception by using try & except block.

Guessing a number program

class ValueTooSmallError (Exception):

 pass

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Niya Towers, Near Satyam Theatre,
Amarpet, Hyderabad-500 016

class ValueTooLargeError (Exception):

 pass

number=10

While True:

 try:

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

 if i_num < number:

 raise ValueTooSmallError

 elif i_num > number:

 raise ValueTooLargeError

 break

except (ValueTooSmallError):

 print("This value is too small try again!")

Except (value `TooLargeError`):

```
    print("This value is too large, try again!")  
    print("Congratulations! you guessed it correctly")  
    print("end")
```

O/P:-

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.

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* To check whether a number is prime or not

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

```
if num > 1:
```

```
    for i in range(2, num):
```

```
        if (num % i) == 0:
```

```
            print(num, "is not a prime number")
```

```
            print(i, "is a factor of", num)
```

```
            break;
```

```
        else:
```

```
            print(num, "is a prime number")
```

```
else:
```

```
    print("num is not a prime number")
```

* To check for factorial

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

```
factorial = 1
```

```
if num < 0:
```

```
    print("Sorry, factorial does not exist for negative numbers")
```

```
elif num == 0:
```

```
    print("The factorial of 0 is 1")
```

```
else:
```

```
    for i in range(1, num+1):
```

```
        factorial = factorial * i
```

```
    print("The factorial of", num, "is", factorial)
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

H16

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* To find the square root of a number

for positive real numbers

```
num = input("Enter a number")
```

```
num_sqrt = num ** 0.5
```

```
print('The square root of {} is {}'.format(num, num_sqrt))
```

for complex numbers:

```
import cmath
```

```
num = 1+2j
```

```
num_sqrt = cmath.sqrt(num)
```

```
print('The square root of {} is {}'.format(num, num_sqrt))
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Saiyam Theatre,
Ameerpet, Hyderabad-500 016

* To check Armstrong number

Armstrong number: if we calculate for 3 digit number then
the addition of the cube of each digit sum is equal to the
corresponding number.

$$\text{Ex.: } 153 \rightarrow 1^3 + 5^3 + 3^3 = 153$$

Program.

```
sum = 0
```

```
temp = num
```

```
while temp > 0:
```

```
    digit = temp % 10
```

```
    sum += digit ** 3
```

```
    temp // = 10
```

```
if num == sum:
```

```
    print(num, "is an Armstrong number")
```

```
else:
```

```
    print(num, "is not an Armstrong number")
```

* To check Armstrong number for n digits

num = 1634

order = len(str(num))

sum = 0

temp = num

while temp > 0:

digit = temp % 10

sum += digit ** order

temp // = 10

if num == sum:

print(f"num, "is an Armstrong number")

else:

print(f"num, "is not an Armstrong number")

* To check Armstrong number within a range

lower = 100

upper = 2000

for num in range(lower, upper + 1):

order = len(str(num))

sum = 0

temp = num

while temp > 0:

digit = temp % 10

sum += digit ** order

temp // = 10

if num == sum:

print(num)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Saiyam Theatre,
Ameerpet, Hyderabad-500 016

* To find factors of a number

def point-factors(x):

 print("The factors of ", x, " are: ")

 for i in range(1, x+1):

 if x % i == 0:

 print(i)

* To find factorial of a number using recursion

def recur-factorial(n):

 if n == 1:

 return n

 else:

 return n * recur-factorial(n-1)

if n < 0:

 print("Sorry, factorial does not exist for numbers")

else num == 0:

 print("Factorial of 0 is 1")

else:

 print("The factorial of ", n, " is ", recur-factorial(n))

* To print fibonacci sequence

Fibonacci Sequence: A fibonacci sequence is the integer sequence of 0, 1, 1, 2, 3, 5, 8 ...

The first two numbers are 0 and 1, by adding two numbers we get 3rd number. All other terms are obtained by adding the preceding two numbers. This means to say nth term is the sum of (n-1)th and (n-2)th terms.

program .

n terms = 10

n1 = 0

n2 = 1

Count = 0

if n terms <= 0:

 print ("please enter a positive integer")

elif n terms == 1:

 print ("fibonacci sequence upto", n terms, ":")

 print (n1)

else:

 print ("fibonacci sequence upto", n terms, ":")

 while Count < n terms:

 print (n1, end = ',')

 n th = n1 + n2

 n1 = n2

 n2 = n th

 Count + = 1

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Note: To test the program, change the value of n terms

first n1 = 0 · n2 = 1

if the num of terms are more than 2, we use a while loop

to find the next term in the sequence by adding the preceding

two terms we then interchange the variables & continue with

the process

* fibonacci sequence using recursive functions

```
def recur_fibon(n):
```

```
    if n<=1:
```

```
        return n
```

```
    else:
```

```
        return (recur_fibon(n-1)+recur_fibon(n-2))
```

```
n_terms = 10
```

```
if n_terms <= 0:
```

```
    print("please enter a positive integer")
```

```
else:
```

```
    print("fibonacci sequence")
```

```
for i in range(n_terms):
```

```
    print(recur_fibon(i))
```

* To print sum of natural numbers

```
num=16
```

```
if num<0:
```

```
    print("Enter a positive number")
```

```
else :
```

```
    sum=0
```

```
    while (num>0):
```

```
        sum += num
```

```
        num -= 1
```

```
    print("The sum is ",sum)
```

* To find LCM

```
def lcm(x,y):
```

```
    if x>y:
```

```
        greater = x
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

else:

 greater=y

while (true):

 if ((greater % 2 == 0) and (greater % y == 0)):

 lcm = greater

 break

 greater += 1

return lcm

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Print ("The Lcm of ", num1, "and", num2, "is", lcm(num1, num2))

using GCD function to find LCM

def gcd(x,y):

 while(y):

 x, y = y, x % y

 return x

def lcm(x,y):

 lcm = (x * y) // gcd(x,y)

 return lcm

Print ("The Lcm of ", num1, "and", num2, "is", lcm(num1, num2))

* To find HCF

def computeHCF(x,y):

 if x > y:

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,

Ameerpet, Hyderabad-500 016

 else:

 smaller = x

 for i in range(1, smaller+1):

 if ((x % i == 0) and (y % i == 0)):

$\text{hcf} = i$

return hcf

print ("The HCF of ", num1, " and ", num2, " is ", computeHCF(num1, num2))

* To calculate the Area of triangle

a = int ("input enter a number")

b = int ("input enter b number")

c = int ("input enter c number")

$$s = (a+b+c)/2$$

$$\text{area} = (s * (s-a) * (s-b) * (s-c))^{1/2} * 0.5$$

print ('The area of triangle is %.2f' % area)

* To solve quadratic equation

import cmath

$$a = 1$$

$$b = 5$$

$$c = 6$$

$$d = (b * * 2) - (4 * a * c)$$

$$\text{sol1} = (-b - \text{cmath.sqrt}(d)) / (2 * a)$$

$$\text{sol2} = (-b + \text{cmath.sqrt}(d)) / (2 * a)$$

print ('The solution are {} and {}'.format(sol1, sol2))

* To swap two variables

$$x = 5$$

$$y = 10$$

$$\text{temp} = 7$$

$$x = y$$

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

y = temp

point ('The value of x after swapping : {}'.format(y))

print ('The value of y after swapping : {}'.format(y))

* To generate a random number

import random

point (random, randint(0, 9))

* To convert kilometers to miles

kilometers = 5.5

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

kilometers = input ("Enter value in kilometers")

conv-fac = 0.621371

miles = kilometers * conv-fac

point ('{} km is equal to {:.2f} miles'.format(kilometers, miles))

* To convert celsius to fahrenheit

celsius = 37.5

fahrenheit = (celsius * 1.8) + 32

point ('{} degree celsius is equal to {:.2f} degree
fahrenheit'.format(celsius, fahrenheit))

* To check a leap year

year = 2000

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

if (year % 4) == 0:

 if (year % 100) == 0:

 if (year % 400) == 0:

 point ("{} is a leap year".format(year))

else:

```
print ("{}04 is not a leap year".format(year))
```

else:

```
point ("Is {0} a leap year?", format(year))
```

else:

```
point("{0} is not a leap year".format(year))
```

* How to convert decimal number into binary, octal, & hexadecimal

dec = 344

```
print ("The decimal value of ", dec, " is: ")
```

```
Print (bin(dec), "in binary")
```

print oct(dec), "in octal")

```
print (hex(dec), "in hexadecimal")
```

* How to convert the ASCII value

$$C = \epsilon P^{-1}$$

point ("The ASCII value of 'c' is", ord(c))

* How to shuffle deck of card

```
import itertools, random
```

```
deck = list(litertools.product(range(1,14),['Spade','Heart','Diamond','Club']))
```

random.shuffle(deck)

```
Print ("you got : ")
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
for i in range(5):
```

```
print(deck[i][0], "of", deck[i][1])
```

* python program to add two matrices

$$x = [[1, 2, 3],\\ [4, 5, 6],\\ [7, 8, 9]]$$

$$y = [[5, 8, 1],\\ [6, 7, 3],\\ [4, 5, 9]]$$

$$\text{result} = [[0, 0, 0],\\ [0, 0, 0],\\ [0, 0, 0]]$$

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

for i in range(len(x)):

 for j in range(len(x[0])):

$$\text{result}[i][j] = x[i][j] + y[i][j]$$

for o in result:

 print(o)

* python program to transpose a matrix

$$x = [[1, 2, 3],\\ [4, 5],\\ [3, 8]]$$

$$\text{result} = [[0, 0, 0],\\ [0, 0, 0]]$$

for i in range(len(x)):

 for j in range(len(x[0])):

$$\text{result}[j][i] = x[i][j]$$

for o in result:

 print(o)

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mira Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* Python program to multiply a matrix

```
x = [[1, 2, 3],  
     [4, 5, 6],  
     [7, 8, 9]]
```

```
y = [[5, 8, 1, 2],  
     [6, 7, 3, 0],  
     [4, 5, 9, 1]]
```

```
result = [[0, 0, 0, 0],  
          [0, 0, 0, 0],  
          [0, 0, 0, 0]]
```

```
for i in range(len(x)):
```

```
    for j in range(len(y[0])):
```

```
        for k in range(len(y)):
```

```
            result[i][j] += x[i][k] * y[k][j]
```

```
for r in result:
```

```
    print(r)
```

* A string is palindrome or not

A palindrome is a string which is same read forward or backwards.

```
my-str = 'aIbohPhoBiA'
```

```
my-str = my-str[::-1]
```

```
rev-str = reversed(my-str)
```

```
if list(my-str) == list(rev-str):
```

```
    print("It is palindrome")
```

```
else:
```

```
    print("It is not palindrome")
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Saiyam Theatre,
Ameerpet, Hyderabad-500 016

* program to sort words in alphabetic order

my-str = "Hello this is an example with cased letters"

words = my-str.split()

words.sort()

print ("The sorted words are: ")

for word in words:

 print (word)

* program to illustrate different set of operations

E = {0, 2, 4, 6, 8}

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

print ("union of E and N is ", E | N)

print ("Intersection of E and N is ", E & N)

print ("Difference of E and N is ", E - N)

print ("symmetric difference of E and N is ", E ^ N)

* python program to count each vowel

vowels = 'aeiou'

ip-str = 'Hello, have you tried our tutorial section yet?'

ip-str = ip-str.casefold()

count = { } .fromkeys(vowels, 0)

for char in ip-str:

 if char in count:

 count[char] += 1

print (count)

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* python program to merge 'mails'

with open("names.txt", 'r', encoding='utf-8') as names_file:

 with open("body.txt", 'r', encoding='utf-8') as body_file:

 body = body_file.read()

 for name in names_file:

 mail = "Hello! " + name + body

 with open(name.strip() + ".txt", 'w', encoding='utf-8')

 as mail_file:

 mail_file.write(mail)

* To find the resolution of the image

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

def jpeg_res(filename):

 """ This function prints the resolution of the jpeg
 image file passed into it """

print("The resolution of the image is ", width, "x", height)

jpeg_res("img1.jpg")

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

OOPS principles:-

OOPS principles are nothing but certain rules or guidelines.

* Different OOPS principle are

1) Encapsulation

2) Inheritance

3) Polymorphism

4) Abstraction.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* If we implement business applications in any programming language according to OOPS principles. Then we will get the security, flexibility and reusability in those applications.

* Object Oriented Programming Languages :-

Any programming language which provides special syntaxes to implement the applications in those languages according to the OOPS principle is known as object oriented programming language.

Examples:-

C++

JAVA

.NET

PYTHON

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* If we implement the business applications in python language according to procedure orientation it looks like as follows.

cname = None
cadd = None
cachno = None
cbal = None
ename = None
eadd = None

eid = None
esal = None

Global Variables.

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

deb deposit ():

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

deb withdraw ():

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

deb transfer ():

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

deb balenq ():

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

deb da ():

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

deb hra ():

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

deb pf ():

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

deb tax ():

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

deb tsal ():

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

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* In procedure orientation mechanism, security is missing for the global variables.

* Encapsulation:

The concept of binding or grouping related data members along with its related functionalities is known as Encapsulation.

Ex:-

cname = None

cadd = None

caeno = None

cbal = None

det deposit ():

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

det withdraw ():

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

det transfer ():

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

det balenq ():

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

ename = None

eadd = None

eid = None

esal = None

det dq ():

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

det hrq ():

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

det pf ():

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

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

def tsal ():

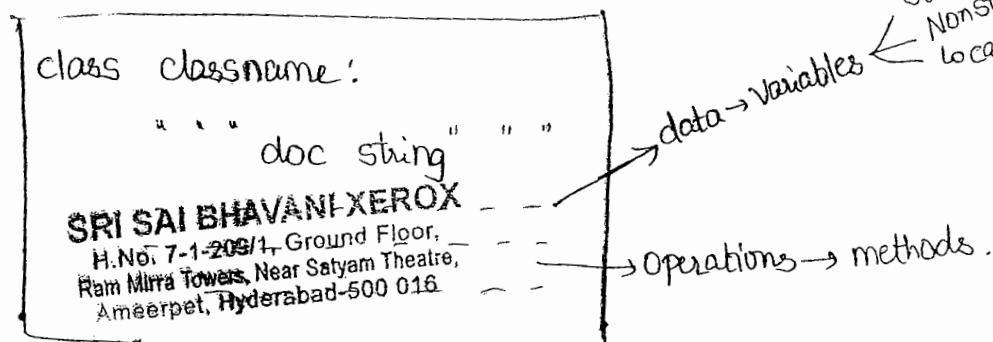
SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Class:-

class is a syntax or structure is used to group the related data members along with its related functionalities.

Syntax:-



- * Within the class we can represent the data by using variables.
- * Within the class we can define the three types of variables. They are:
 - 1) Static variables
 - 2) Non static variables
 - 3) Local variables
- * Within the class we can represent the operations by using methods.
- * Documentation string is used to provide the description about the class.
- * Documentation string is optional within the class syntax.
- * If u create an object if the object is created with original address. that variable is called as pointer variables.
- * If you create an object if the object is created with indirect address of the object is called reference variables. Python interpreter is writing with the original addresses of the object internally.

Object:-

The concept of allocating memory space for members of a class at run time dynamically is known as a object.

Syntax:-

Reference variable = classname() → object creation statement.

- * After creating the object, python interpreter internally takes the original addresses of the object, creates the indirect addresses based on the original addresses of the object and gives that indirect addresses to variables.
- * The variable which is storing indirect address of the object is known as reference variable.
- * Through the reference variable, we can get the data into the object, we can call the methods on the objects.
- * We can create 'n' no of objects for a class.
- * Two different objects are same class or different classes doesn't contain same indirect address.

Example:-

```
print("begin")
class X:
    def m1(self):
        print("in m1 of X")
    def m2(self):
        print("in m2 of X")
```

x1=x()

print(x1)

x1.m1()

x1.m2()

x1.m1()

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
x2 = x()
print(x2)
x2.m1()
x2.m2()
print("end")
```

O/P:-

```
begin
<-- main --> x object at 0x01FECC10>
in m1 of x
in m2 of x
in m1 of x
<-- main --> x object at 0x02A9C10>
in m1 of x
in m2 of x
end.
>>>
```

Static Variables (Class variables) :-

The variables which are declared within the class outside of all methods are known as 'static variables'.

- * The data which is common for all the objects is recommended to represent through static variables.
- * Static variables of a class, memory will be allocated only once.
- * Static variables of a class, we can access within all the methods of a class through class name.
- * static variables of a class we can access outside of the class by using class name.
- * If we apply abstraction on static variable we can only access static variables inside of the class not outside of the class.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Example:- class X :

a = 1000

def m1(self):

 print(x.a)

x1 = X()

x1.m1()

print(x.a)

x2 = X()

x2.m1()

O/p:-

1000

1000

1000

>>>

Example:- class X :

a = 1000

b = 2000

def display(self):

 print(x.a)

 print(x.b)

def modify(self):

 x.a = 3000

 x.b = 4000

x1 = X()

x1.display()

x1.modify()

x1.display()

x2 = X()

x2.display()

O/p:-

1000

2000

3000

4000

3000

4000

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Non-static Variables (Instance Variables)

The variables which are declared within the class by using (self.) are known as non-static variables.

- * The data which is separate for every is recommended to represent through non static variables.
- * Non static variables of a class memory will be allocated within the object.
- * Non static variables of a class we can access within the same class by using self.
- * Non static variables of a class we can access outside of the class by using 'reference variable'.

Example:-

Class X :

def m1(self) :

 self.a = 1000

 self.b = 2000

def m2(self) :

 print(self.a)

 print(self.b)

x1 = X()

x1.m1()

x1.m2()

print(x1.a)

print(x1.b)

x2 = X()

x2.m1()

x2.m2()

O/P:- 1000

2000

1000

2000

1000

2000

→→→

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Example:- class X :

def m1(self):

self.a = 1000

self.b = 2000

def display(self):

print(self.a)

print(self.b)

def modify(self):

self.a = 3000

self.b = 4000

x1 = X()

x1.m1()

x1.display()

x1.modify()

x1.display()

x2 = X()

x2.m1()

x2.display()

O/P :-

1000

2000

3000

4000

1000

2000

>>>

Whenever we define non static variables within the methods those non static variables will be added to the object whenever we call that method.

Example:- class X :

def m1(self):

self.a = 1000

self.b = 2000

def m2(self):

print(self.a)

print(self.b)

SRI SAI BHARATHI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHARATHI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

x1=x()
x1.m1()
x1.m2()
x2=x()
x2.m2()

O/p:- 1000
2000

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Attribute Error:- 'x' object has no attribute 'a'.

Constructors:-

- * Constructor is a special method.
- * Constructor name should be `--init--(self)`
- * Constructor is used to define & initialize non static variables.

Differences b/w methods & constructors

Methods

Constructor

- | | |
|---|--|
| 1) method name can be any name | 1) constructor name should be <code>--init--</code> |
| 2) Method will be executed whenever we call a method. | 2) constructor will be executed automatically whenever we create a object. |
| 3) With respect to one object one method can be called for 'n' members of lines | 3) With respect to one object one constructor can be executed only once. |
| 4) Methods are used to represent business logic to perform the operations. | 4) Constructors are used to define & initialize the non static variable. |

Example:-

class X:

```
def __init__(self):  
    print("in const of x")  
  
def m1(self):  
    print("in m1 of x")
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

$x1 = x()$
 $x1.m1()$
 $x1.m1()$
 $x2 = x()$
 $x2.m1()$
 $x2.m1()$

O/P: -
in const of x
in m1 of x
in m1 of x
in const of x
in m1 of x
in m1 of x

Example: - class x :

```
def __init__(self):  
    self.a = 1000  
    self.b = 2000  
def display(self):  
    print(self.a)  
    print(self.b)  
def modify(self):  
    self.a = self.a + 100  
    self.b = self.b + 200
```

$x1 = x()$
 $x1.display()$
 $x1.modify()$
 $x1.display()$
 $x2 = x()$
 $x2.display()$

O/P: -
1000
2000
1100
2200
1000
2000
222

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

def __init__(self a, b):

self.a = a

self.b = b

def display(self):

print(self.a)

print(self.b)

def modify(self):

self.a = self.a + 100

self.b = self.b + 200

X1 = X(1000, 2000)

X1.display()

X1.modify()

X1.display()

X2 = X(3000, 4000)

X2.display()

X2.modify()

X2.display()

O/P:-

1000

2000

1100

2200

3000

4000

3100

4200

>>>

Local Variables:-

The variables which are declared within the method body are known as local variables.

* The data which is required to use within a particular method is recommended to represent through local variables.

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,

Ram Mirra Towers, Near Satyam Theatre,

Ameerpet, Hyderabad-500 016

- * For local variables of a method memory will be unoccupied whenever we make a method call.
- * Local variables of a method can't be accessed outside of a method.

Ex:- class X :

```
def m1(self):
    a = 1000
```

```
    print(a)
```

```
def m2(self):
    b = 2000
```

```
    print(b)
```

```
x1 = X()
```

```
x1.m1()
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
x1.m2()
```

```
x1.m1()
```

O/P:-

```
1000
```

```
2000
```

```
1000
```

```
>>>
```

Ex:- class Cust :

```
" " " cust app " " "
```

```
cbname = "sbi"
```

```
def __init__(self, cname, cadd, cacno, cbal):
```

```
    self.cname = cname
```

```
    self.cadd = cadd
```

```
    self.cacno = cacno
```

```
    self.cbal = cbal
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor, self.cbal = cbal
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

deposit(self, damt):

```
    self.cbal = self.cbal + damt
```

```
def withdraw(self, wamt):
```

```
    self.cbal = self.cbal - wamt
```

```
def display(self):
```

```
    print(self.cname)
```

print (self, cadd)
print (self, cacho)
print (self, cbal)
print (cust, cbname)

c1 = cust ("ravana", "colombo", 1001, 100000.00)
print (c1)

c1. deposit (20000.00)

c1. withdraw (40000.00)

c1. display ()

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre.
Ameerpet, Hyderabad-500 016

c2 = cust ("sita", "hyd", 1002, 1000.00)

print (c2)

c2. deposit (3000.00)

c2. withdraw (2000.00)

c2. deposit (4000.00)

c2. display ()

Op:- <-- main --. cust object at 0X023A9D50>

ravana

colombo

1001

80000.0

sbi

<-- main --. cust object at 0X01FFCD50>

sita

hyd

1002

6000.0

sbi

>>>

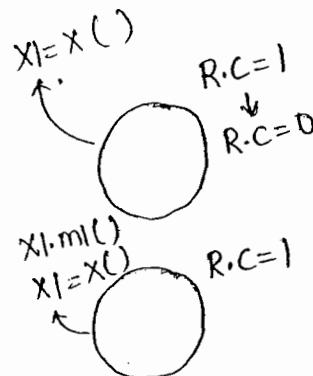
SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre.
Ameerpet, Hyderabad-500 016

Garbage Collection:-

- * The concept of removing unused or unreferenced objects from the memory location is known as a Garbage collection.
- * While executing the program, if garbage collection takes place then more memory space is available for the program and rest of the program execution becomes faster.
- * Garbage collector is a predefined program, which removes the unused or unreferenced objects from the memory location.
- * Any object reference count becomes zero then we call that object as a unused or unreferenced object.
- * Then no. of reference variables which are pointing the object is known as a reference count of the object.
- * While executing the python program if any object reference count becomes zero, then internally python interpreter calls the garbage collector and garbage collector will remove that object from memory location.

Ex:-

```
class X:  
    def m1(self):  
        print ("in m1 of X")
```



When we are creating the same object it removes the first address of $x1$ and then place second address of $x1$. Now R.c becomes zero, the python interpreter calls the garbage collector & it removes that object from memory location.

Destructor:-

Destructor is one special method and the destructor name should be "`-- del --`".

* Destructor will be executed automatically, whenever object is deleting from memory location.

Ex:- class X:

```
def __init__(self):  
    print("in constructor of X")  
  
def m1(self):  
    print("in m1 of X")  
  
def __del__(self):  
    print("in destructor of X")
```

X1 = X()

print(X1)

X1.m1()

X1 = X()

print(X1)

X1.m1()

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- in constructor of X

<__main__.X object at 0X023A8DD0>

in m1 of X

in constructor of X

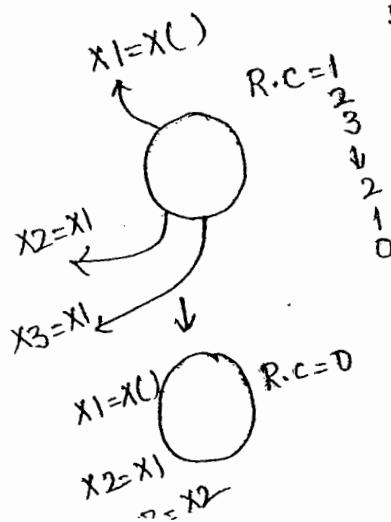
in destructor of X

<__main__.X object at 0XD1FDC0D0>

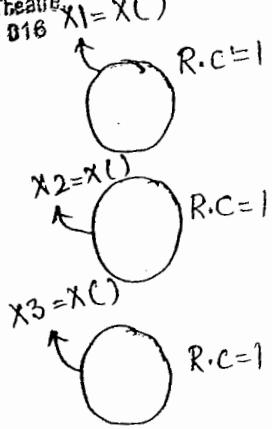
in m1 of X

>>>

Ex:-



SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre
Ameerpet, Hyderabad-500 016



Program :- Class X :

def __init__(self):

 print ("in constructor of x")

def __del__(self):

 print ("in destructor of x")

x1 = X()

print(x1)

x2 = x1

print(x2)

x3 = x2

print(x3)

x1 = X()

print(x1)

x2 = x()

print(x2)

x3 = x()

print(x3)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- in constructor of x

<__main__.X object at 0x023A8DD0>

<__main__.X object at 0x023A8DD0>

<__main__.X object at 0x023A8DD0>

in constructor of x

<__main__.X object at 0XD1FCCD70>

in constructor of x

<__main__.X object at 0X01FF5150>

in constructor of x

in destructor of x

<__main__.X object at 0X023DF9B0>

>>>

* del removes the variables but not the object, it decreases
the reference count

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Q:-

- * Del is a keyword, which removes given variable or variables from the memory location.
- * Whenever del removes a variable the reference count of the object which is pointed by that variable will be decreased by one.

Ex:- class X:

```
def __init__(self):  
    print("* in constructor of X")  
  
def __del__(self):  
    print("* in destructor of X")
```

x1 = X()

print(x1)

x2 = x1

print(x2)

x3 = x2

print(x3)

del x1

del x2

del x3

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- in constructor of X

```
<__main__.X object at 0X023A8DDD>  
<__main__.X object at 0X023A8DDD>  
<__main__.X object at 0X023A8DDD>
```

in destructor of X

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- class Emp :

 " " " emp app" " "

 empcount = 0

 def __init__(self, ename, eadd, eid, esal):

 self.ename = ename

 self.eadd = eadd

 self.eid = eid

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,

Ram Mirra Towers, Near Satyam Theatre,

Ameerpet, Hyderabad-500 016

 self.esal = esal

 Emp.empcount = Emp.empcount + 1

 def display(self):

 print(self.ename, self.eadd, self.eid, self.esal)

 del __del__(self):

 Emp.empcount = Emp.empcount - 1

e1 = Emp("scott", "dallas", 7788, 3000.00)

e1.display()

e2 = Emp("blake", "mumbai", 7902, 4000.00)

e2.display()

e3 = Emp("smith", "hyd", 7369, 3500)

e3.display()

print("Total employees are:", Emp.empcount)

del e1

print("Total employees are:", Emp.empcount)

O/P:-

| | | | |
|-------|--------|------|---------|
| scott | dallas | 7788 | 3000.00 |
| blake | mumbai | 7902 | 4000.00 |
| smith | hyd | 7369 | 3500.0 |

Total employees are: 3

Total employees are: 2

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,

Ram Mirra Towers, Near Satyam Theatre,

Ameerpet, Hyderabad-500 016

>>>

Ex:- class X:

```
def __init__(self):
```

```
    print("in constructor of X")
```

```
def __del__(self):
```

```
    print("in destructor of X")
```

```
lst = [X(), X(), X(), X()]
```

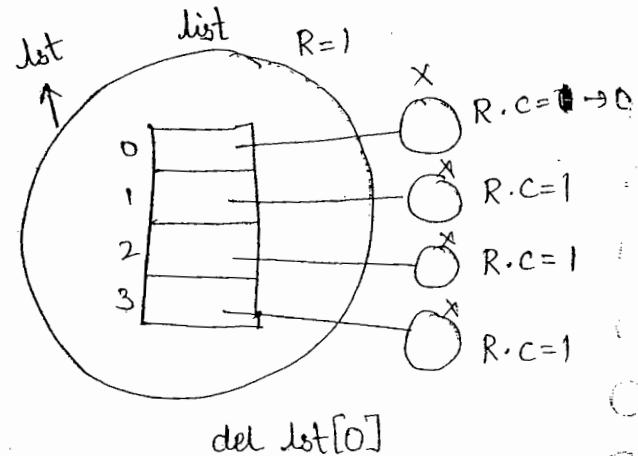
```
del lst[0]
```

```
del lst
```

O/P:-

in constructor of X

del lst[0]



Adding static variables to the class from outside of the class:-

* We can add the static variables to the class from outside of the class by using class name

Ex:- class X:

```
a = 1000
```

```
def display(self):
```

```
    print(x.a)
```

```
x1 = X()
```

```
x1.display()
```

```
x.b = 2000
```

```
print(x.a)
```

```
print(x.b)
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/p:-

1000
1000
2000
 \ggg

Removing static variables of a class from outside of the class.

* We can delete the static variables of a class from outside of the class by using following syntax.

`del classname.staticvariable`

Ex:- class X :

a = 1000

b = 2000

def display(self):

 print(x.a) **SRI SAI BHAVANI XEROX**

 print(x.b)

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

x1 = X()

x1.display()

del x.b

x2 = X()

x2.display()

O/p:-

2000

1000

Attribute Error.

Adding non static variables to the object from outside of the class

* We can add the nonstatic variables to the objects from outside of the class by using reference variable.

* The nonstatic variables which are added to one object will not be available in another object.

class X :

def __init__(self):

self.a = 1000

self.b = 2000

def display(self):

print(self.a)

print(self.b)

x1 = X()

x1.display()

x1.c = 3000

print(x1.a)

print(x1.b)

print(x1.c)

x2 = X()

print(x2.a)

print(x2.b)

print(x2.c)

O/P:- 1000

2000

1000

2000

3000

1000

2000

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Attribute error: 'x' has no attribute 'c'

Removing nonstatic variables of a object from outside of the class

* We can remove nonstatic variables of a object from outside of the class by using following syntax.

[def reference variable name.non static variable]

* The nonstatic variables which are deleted from one object will not be deleted from another objects

Ex:- class X:

def __init__(self):

 self.a = 1000

 self.b = 2000

def display(self):

 print(self.a)

 print(self.b)

x1 = X()

x1.display()

del x1.b

x2 = X()

x2.display()

O/P:-

1000

2000

1000

2000

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Using properties of one class into another class:-

* We can use the properties of one class into another class in two ways. They are

① Has a relationship

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

② Is a relationship

① Has a relationship:- The concept of using the properties of

one class into another class by using class name or by

using reference variable name is known as a

"Has a relationship".

Ex:- class X :

a = 1000

def __init__(self):

self.b = 2000

def m1(self):

print("in m1 of x")

class Y:

c = 3000

def __init__(self):

self.d = 4000

def m2(self):

print("in m2 of y")

def m3(self):

print(x.a)

x1 = X()

.print(x1.b)

x1.m1()

.print(y.c)

.print(self.d)

self.m2()

print("in m3 of y")

y1 = Y()

y1.m3()

O/P:-

1000

2000

in m1 of x

3000

4000

in m2 of y

in m3 of y

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

IS a relationship (or) Inheritance :-

- * The concept of using the properties of one class into another class directly is known as inheritance (or) IS a relationship.
- * A class which is extended by another class is known as a superclass or base class.
- * A class which is extending another class is known as a subclass or derived class.
- * Superclass properties directly we can access into the subclass like as a subclass members

Ex:- class X :

a = 1000

def m1(self):

 print("in m1 of x")

class Y(X):

c = 3000

def __init__(self):

 self.d = 4000

def m2(self):

 print("in m2 of y")

def m3(self):

 print(y.a)

 self.m1()

 print(y.c)

 print(self.d)

 self.m2()

 print("in m3 of y")

y1 = Y()

y1.m3()

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:-

1000
in m1 of X
3000
4000
in m2 of Y
in m3 of Y
>>>

* Both superclass properties & subclass properties we can access through subclass name or through super variable.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- Class X:

a=1000

def m1(self):

 print("in m1 of X")

class Y(X):

c=3000

def __init__(self):

 self.d=4000

def m2(self):

 print("in m2 of Y")

print(y.a)

print(y.c)

y1=y()

print(y1.d)

y1.m1()

y1.m2()

O/P:-

1000
3000
4000
in m1 of X
in m2 of Y
>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- class X:

```
def __init__(self):
```

```
    print("in const of X")
```

```
def m1(self):
```

```
    print("in m1 of X")
```

class Y(X):

```
def m2(self):
```

```
    print("in m2 of Y")
```

y1 = Y()

y1.m1()

y1.m2()

Output: - in const of X

in m1 of X

in m2 of Y.

Object class:-

* Object class is a predefined class, which is defined in built-in-modules.

* Object class is a superclass for every class in python.

* Object class properties we can use directly in every class.

* Object class properties we can access through every class reference variable.

Ex:- class X:

```
def m1(self):
```

```
    print("in m1 of X")
```

class Y(X):

```
def m2(self):
```

```
    print("in m2 of Y")
```

```
print(y.__bases__)
```

```
print(x.__bases__)
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
y1 = y()
y1.m1()
y1.m2()
a = y1.__hash__()
print(a)
```

O/p:- (<class '__main__.X'>,)

(<class 'object'>,)

In m1 of X

In m2 of y

2087443

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- class X:

def m1(self):

print("In m1 of X")

x1 = X()

print(x1)

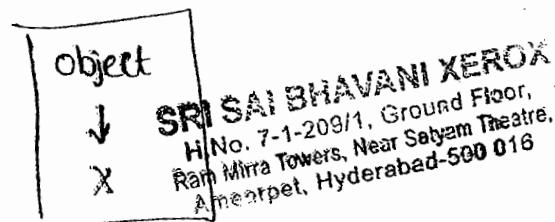
a = x1.__str__()

print(a)

O/p:- <__main__.X object at 0xb23ac190>
<__main__.X object at 0xb23ac190>
>>>

Types of inheritances:-

Single Inheritance:- The concept of inheriting the properties from only one class into another class is known as a single inheritance



Ex:- class X(object):

def m1(self):

print("in m1 of X")

x1=X()

x1.m1()

a=x1.__hash__()

print(a)

O/P:-

in m1 of X

2337815

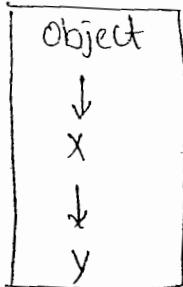
>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Niya Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Multi level Inheritance:-

* The concept of inheriting the properties from multiple classes into single class with the concept of one after another is known as a multi level inheritance.



Ex:- class X(object):

def m1(self):

print("in m1 of X")

class Y(X):

def m2(self):

print("in m2 of Y")

y1=Y()

y1.m1()

y1.m2()

a=y1.__hash__()

print(a)

O/P:-

in m1 of X

in m2 of Y

2337815

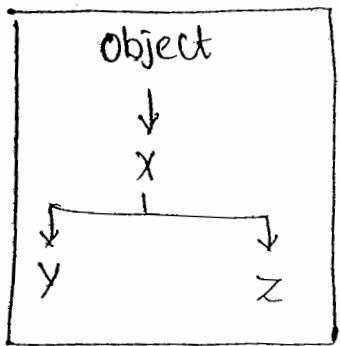
>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Niya Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Hierarchical Inheritance :-

- * The concept of inheriting the properties from one class into multiple classes separately is known as hierarchical inheritance.



Ex:-

class x(object):
 def m1(self):
 print("in m1 of x")
 class y(x):
 def m2(self):
 print("in m2 of y")
 class z(x):
 def m3(self):
 print("in m3 of z")
 y1 = y()
 y1.m1()
 y1.m2()
 a = y1.__hash__()
 print(a)
 z1 = z()
 z1.m1()
 z1.m3()
 b = z1.__hash__()
 print(b)

O/p:-

in m1 of x

in m2 of y

2337815

in m1 of x

in m3 of z

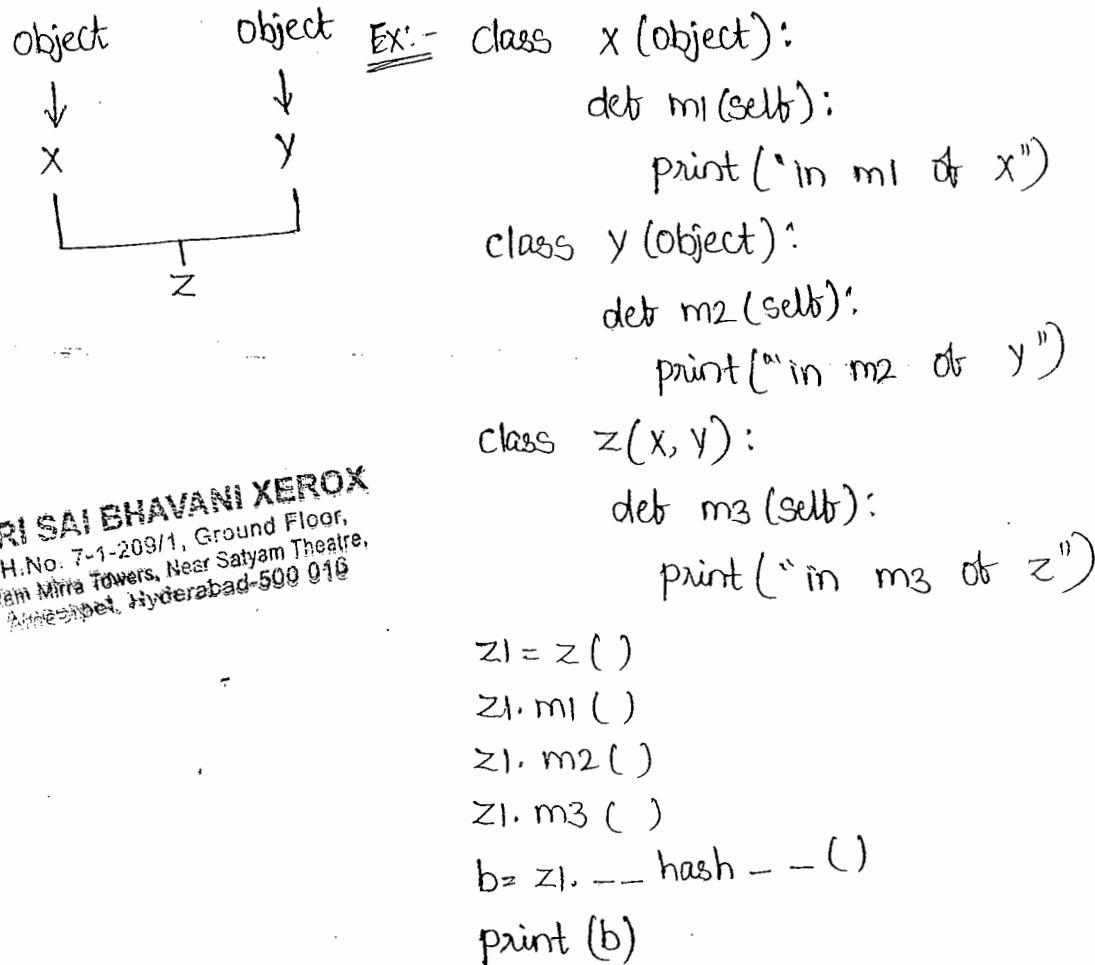
2099735

>>>

SRI SAI BHAVANI XEROX
 H.No. 7-1-209/1, Ground Floor,
 Ram Mirra Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

Multiple Inheritance:-

* The concept of inheriting the properties from multiple classes into single class at a time is known as multiple inheritance.



Op:- in m1 of x
 in m2 of y
 in m3 of z
 2266095.

Ex:- class X:

def m1(self):
 print("in m1 of X")

class Y:

def m1(self):
 print("in m1 of Y")

class Z(X, Y):

def m2(self):

SRI SAI BHAVANI XEROX
 H.No. 7-1-209/1, Ground Floor,
 Ram Mirra Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

print("in m2 of z")

z1 = z()

z1.m1()

z1.m2()

O/p:- in m1 of X

in m2 of z.

>>>

Cyclic Inheritance:-

* The concept of inheriting the superclass is known as ~~SRI SAI BHAVANI XEROX~~ 'cyclic inheritance'.

* Cyclic inheritance is not supported by Python.

Ex:- class X(z):

```
def m1(self):
```

```
    print("in m1 of X")
```

class Y(X):

```
def m2(self):
```

```
    print("in m2 of Y")
```

class Z(Y):

```
def m3(self):
```

```
    print("in m3 of Z")
```

z1 = z()

z1.m1()

z1.m2()

z1.m3()

O/p:- Error



SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Saiyam Theatre,
Ameerpet, Hyderabad-500 016

Hybrid Inheritance:-

* Combination of single, multilevel, hierarchical and multiple inheritances is known as a hybrid inheritance.

Polymorphism:-

* The concept of defining multiple functionalities or multiple logics to perform the same operations is known as a polymorphism.

* poly means 'many', morphism means 'forms', forms means functionalities or logic.

* Polymorphism can be implemented through method overloading or method overriding syntaxes.

Overloading:-

* The concept of defining multiple methods with the same name or same no. of parameters, different no. of parameters within a class is known as a method overloading.

Ex:- Class X:

```
def m1(self):
    print("in no param m1 of X")
def m1(self, a):
    print("in one param m1 of X")
def m1(self, a, b):
    print("in two param m1 of X")
```

```
x1=X()
x1.m1(100,200)
```

Op:- in two param m1 of X

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX,
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* Whenever we define multiple methods with same name within a class then by default python interpreter recognizes 'last defined method' only

Ex:- class X:

```
def m1(self):  
    print("in SRI SAI BHAVANI XEROX  
H.No. 7-1-209/1, Ground Floor,  
Ram Mirra Towers, Near Satyam Theatre,  
Ameerpet, Hyderabad-500 016")  
def m1(self, a)  
    print("in one param m1 of X")  
def m1(self, a, b)  
    print("in two param m1 of X")
```

```
x1=X()  
x1.m1(1000)
```

O/P:- error

Ex:- class X:

```
def m1(self):  
    print("in second m1 of X")  
def m1(self):  
    print("in first m1 of X")
```

```
x1=X()  
x1.m1()
```

O/P:- in first m1 of X

Ex:- Constructor Overloading:-

class X:

```
def __init__(self, a):  
    self.a=a  
def __init__(self, b, c):  
    self.b=b  
    self.c=c
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
x1 = X(100, 200)
print(x1.b)
print(x1.c)
print(x1.a)
```

O/P:-
100
200

Error it is not accessing variable 'a'

Ex:- class X:

```
def __init__(self, a):
```

```
    self.a = a
```

```
def __init__(self, b, c):
```

```
    self.b = b
    self.c = c
```

```
X1 = X(100, 200)
```

O/P:- Error

Ex:- class X:

```
def add(self, instanceOf, *args):
```

```
    if instanceOf == 'init':
```

```
        result = 0
```

```
    if instanceOf == 'str':
```

```
        result = ''
```

```
    for i in args:
```

```
        result = result + i
```

```
    print(result)
```

```
X1 = X()
```

```
X1.add('init', 10, 20, 30)
```

```
X1.add('str', 'sathya', 'tech')
```

O/P:-

60
sathyatech

This is not supporting overloading.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Method Overriding :- The concept of defining multiple methods with the same name, with same no. of parameters or different no of parameters one is in super class, another one is in subclass is known as method overriding.

* Whenever we override superclass method in subclass then by default subclass method will be executed, if subclass object is created

* We can also access the superclass method after overriding that method in subclass by using `super().` statement.

Ex:- class X:

```
def m1(self):  
    print("in no param m1 of X")
```

class Y(X):

```
def m1(self):  
    print("in no param m1 of Y")
```

y1=y()

y1.m1()

x1=x()

x1.m1()

Op:- In no param m1 of Y
In no param m1 of X.

Ex:- class X:

```
def m1(self):  
    print("in no param m1 of X")
```

class Y(X):

```
def m1(self):  
    print("in no param m1 of Y")
```

def m2(self):

```
self.m1()  
super().m1()
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

print (" in no param m2 of y")

y1 = y()

y1.m1()

O/P:- in no param m1 of y

in no param m1 of x

in no param m2 of y.

With constructor method

Ex:- class X:

def __init__(self):
 self.a = 1000
 self.b = 2000

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

class Y(X):

def __init__(self):
 super().__init__()
 self.c = 3000
 self.d = 4000

y1 = y()
print(y1.a)
print(y1.b)
print(y1.c)
print(y1.d)
x1 = X()
print(x1.a)
print(x1.b)

O/P:- 1000
2000
3000
4000
1000
2000
>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- class X(object):
 def m1(self):
 print("in m1 of x")
 def __str__(self):
 return "sathya".

x1 = X()

print(x1)

a = x1. __str__()

print(a)

O/P:- sathya

sathya

>>>

Ex:- class X(object):

 def __init__(self, msg):

 self.msg = msg

 def m1(self):

 print("in m1 of x")

 def __str__(self):

 return self.msg

x1 = X("sathya")

print(x1)

x2 = X("lokesh")

print(x2)

a = str("python")

print(a)

b = str("django")

print(b)

O/P:-

sathya

lokesh

python

django

-->

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- class cust:

 " " " cust app" " "

 cbname = "sbi"

def __init__(self, cname, cadd, cacno, cbal):

 self.cname = cname

 self.cadd = cadd

 self.cacno = cacno

 self.cbal = cbal

def __str__(self):

 return "self.cname + " + self.cadd + " " + str(self.cacno)
 " " str(self.cbal) + " " + cust.cbname + " ".

def deposit(self, damt):

 self.cbal = self.cbal + damt

def withdraw(self, wamt):

 self.cbal = self.cbal - wamt

def display(self):

 print(self.cname)

 print(self.cadd)

 print(self.cacno)

 print(self.cbal)

 print(self.cbname)

c1 = cust("ravana", "colombo", 1001, 100000.00)

print(c1)

c2 = cust("sita", "hyd", 1002, 1000.00)

print(c2)

c3 = cust("miller", "dallas", 1003, 3000.00)

print(c3)

x = [c1, c2, c3]

for p in x:

 print(p)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

`y=sorted(x, key=lambda c : c.cbal, reverse=True)`
for q in y:
 print(q)

O/P:-

| name | city | id | balance | branch |
|--------|---------|------|-----------|---|
| ravana | colombo | 1001 | 100000.00 | sbi |
| sita | hyd | 1002 | 1000.0 | sbi |
| miller | dallas | 1003 | 3000.0 | SRI SAI BHAVANI XEROX H.No. 7-1-209/1, Ground Floor, Rani Mirra Towers, Near Satyam Theatre, Ameerpet, Hyderabad-500 016 |
| ravana | colombo | 1001 | 100000.00 | sbi |
| sita | hyd | 1002 | 1000.00 | sbi |
| miller | dallas | 1003 | 3000.00 | sbi |
| ravana | colombo | 1001 | 100000.00 | sbi |
| miller | dallas | 1003 | 3000.0 | sbi |
| sita | hyd | 1002 | 1000.0 | sbi |

Abstraction

- * The concept of hiding the properties of a class is known as abstraction.
- * We can hide the properties of a class by using `--`
- * Hidden properties of a class cannot be accessed directly from outside of the class.

Eg:- class X:

-- a=1000

def mi(self):

 print(X.--a)

X1=X()

X1.mi()

print(X.--a)

O/P:- 1000

Attribute Error

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Implementing Inheritance:-

* Hidden properties of a class cannot be accessed directly within its subclass also.

Eg:- class X:

$\underline{\underline{a}} = 1000$

def m1(self):

 print("in m1 of X")

class Y(X):

$\underline{\underline{b}} = 2000$

def m2(self):

 print("in m2 of Y")

def m3(self):

 print(y. $\underline{\underline{a}}$)

 self.m1()

 self.m2()

 print(y. $\underline{\underline{a}}$)

y1 = Y()

y1.m3()

O/p:- 2000

 in m1 of X

 in m2 of Y

Attribute error

>>>

* Hidden properties of a class can be accessed from outside of the class by using some special syntax.

Eg:- class X:

$\underline{\underline{a}} = 1000$

def m1(self):

 print("in m1 of X")

x1 = X()

print("in m1 of X")

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/p:- 1000
>>

Eg:- class X :

-- a = 1000

def m1(self) :

print ("in m1 of x")

class Y(X) :

-- b = 2000

def m2(self) :

print ("in m2 of y")

def m3(self) :

print (y. -- b)

self. m1()

self. m2()

print (self. -- x -- a)

y1 = Y()

y1. m3()

O/p:- 2000

in m1 of x

in m2 of y

1000

>>>

Eg:- class X :

def m1(self) :

print (self)

print ("in m1 of x")

def m2(self, a) :

print (self)

print (a)

print ("in m2 of x")

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
x1=x()
print(x1)
x1.m1()
x1.m2(1000)
x2=x()
print(x2)
x2.m1()
x2.m2(2000)
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Amarpet, Hyderabad-500 016

O/P:-

```
<-- main --> x object at 0X0219DCB0>
<-- main --> x object at 0X0219DCB0>
in m1 of x
<-- main --> x object at 0X0219DCB0>
1000
in m2 of x
<-- main --> x object at 0X0221E090>
<-- main --> x object at 0X0221E090>
in m1 of x
<-- main --> x object at 0X0221E090>
2000
in m2 of x
>>>
```

- * We also call the methods of a class by using class name.
- * At the time of calling the method by using classname we should pass the value to the self parameter of a method.
- * Through class name also methods can be called.

class x :

```
def m1(self):
    print(self)
    print("in m1 of x")
def m2(self,a)
    print(self)
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Amarpet, Hyderabad-500 016

print(a)

print("in m2 of x")

x1=x()

print(x1)

x.m1(x1)

x.m2(x1, 1234)

x2=x()

print(x2)

x.m1(x2)

x.m2(x2, 5678)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram-Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Output:-

<-- main --> Object at 0X012BAC3D>
<-- main --> Object at 0X012BAC3D>
in m1 of x
<-- main --> object at 0XD12BAC3D>
1234
in m2 of x
<-- main --> object at 0X0221E0DD0>
<-- main --> object at 0X0221E0DD0>
in m1 of x
<-- main --> object at 0X0221E0DD0>
5678
in m2 of x

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

PYTHON - DATABASE :-

- * Object Relational data base management system (ORDBMS) are softwares, which stores the data in permanent manner.
- * Object Relational data base management system allows to perform the 'random read' and 'random write' operations on the data of the ORDBMS.
- * ORDBMS provides the security for the data.
- * For online transaction processing (OLTP) operations implementation purpose we have to use ORDBMS.
- * Different ORDBMS are

ORACLE

MYSQL

DB2

SALITE 3

SQLSERVER

TERA DATA

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * To provide the communication b/w python program to the corresponding ORDBMS, we use databases related external modules.
- * Databases related external modules we can install by using

"PIP" application

WWW.oracle.com

10G } Enterprise Edition
11G }
12C }

- * Download & install the oracle database 11g or 12c Enterprise edition.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* Username = SCOTT
password = tiger
ServiceName (or) logical database name (or) SID or host storing
name = orcl

port no = 1521

IP address = local host

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * Download & install "python 2.7 version"
- * "microsoft visual c++ compiler for python 2.7 version" download and install
- * Download & install the "cx-oracle" module by using following command

Search in command prompt

C:\Users\lokesh>cd . . .

C:\Users>cd . . .

C:\>cd python 27

C:\python27>cd scripts

C:\python27\scripts>pip install cx-oracle.

- * We can establish connection with the database by calling connect function of cx-oracle module

Syntax:-

```
import cx-oracle
```

con = cx_oracle.connect ('dbusername', 'dbpwd',
ipaddr of the computer where database is installed : portno /
database service name')

- * Connect function establish the connection with database with the given details, store the connection details into connection object and that object address will be stored into the given variable

- * After establishing the connection with database to sent the sql queries to the database we have to create "cursor object".
- * Cursor Object can be created by calling cursor method of the connection object.

Syntax:-

```
cur = con.cursor()
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * By calling method of cursor objects, we can or execute the sql queries or psql programs.
- * By calling the methods of cursor objects we can read the data from the cursor object which is given by the database (sql) queries or psql programs.
- * After reading the data from the cursor object we have to close the cursor object by calling close method of cursor object.
- * After closing the cursor object we have to close the connection object by calling close method of connection object.

Ex:-

```
import cx_Oracle
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
cur = con.cursor()
cur.execute("select * from dept")
for row in cur:
    print(row)
cur.close()
con.close()
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- dp:-
- (10, 'ACCOUNTING', 'NEWYORK')
 - (20, 'RESEARCH', 'DALLAS')
 - (30, 'SALES', 'CHICAGO')
 - (40, 'OPERATIONS', 'BATSON')

- * Whenever connection is established with the database connection should get closed before terminating the program.
- * To close the connection if the connection is established we use exception handling techniques.

Ex:- import cx_Oracle

con = None

try :

```
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
```

```
print("connection is established")
```

```
cur = con.cursor()
```

```
cur.execute("select * from dept123")
```

```
rows = cur.fetchmany()
```

```
for row in rows:
```

```
    print(row)
```

```
cur.close()
```

```
except(cx_Oracle.DatabaseError):
```

```
    print("error occurred")
```

Finally :

```
con != None :
```

```
    con.close()
```

```
print("connection is closed")
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:- connection is established

error occurred

connection is closed

>>>

EX:- import cx_Oracle

```
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
```

```
cur = con.cursor()
```

```
cur.execute("select empno, ename, sal from emp order by sal  
desc")
```

```
recs = cur.fetchmany(5)
```

```
for rec in recs:
```

```
    print(rec)
```

```
cur.close()
```

```
con.close()
```

O/P:- (7839, 'KING', 5000.0)

(7902, 'FORD', 3000.0)

(7788, 'SCOTT', 3000.0)

(7566, 'JONES', 2975.0)

(7698, 'BLAKE', 2850.0)

EX:- import cx_Oracle

```
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
```

```
cur = con.cursor()
```

```
cur.execute("select empno, ename, sal from emp order by  
sal desc")
```

```
print(cur.fetchone())
```

```
print(cur.fetchone())
```

```
cur.close()
```

```
con.close()
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Q1P:- (7839, 'KING', 5000.0)

(7902, 'FORD', 3000.0)

Executing dml commands through python programs:-

- * Dml commands are used to modify the data of the database objects.
 - * Whenever we execute dml commands, the records are going to be modified temporarily
 - * Whenever we run "roll back" command the modified records will come back to its original state
 - * To modify the records of the database objects permanently we use "commit" command.
 - * After executing the commit command even though we execute "rollback" command, the modified records will not come back to its original state
 - * Create the emp1 table in the database by using following command
- Create table emp1 as select * from emp;
- * Whenever we run the dml command through the python program, then the no. of records which are modified because of that command will be stored into the rowcount attribute of cursor object
 - * After executing the dml command through the python program we have to call commit method of cursor object.

Ex:- import cx_Oracle

```
db = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
```

```
cursor = db.cursor()
```

```
cursor.execute("delete from emp1 where deptno=10")
```

```
print("total no of deleted records are:", cursor.rowcount)
```

```
cursor.close()
```

```
db.commit()
```

```
db.close()
```

Op:- ('total no of deleted records are: 3')

Transaction management:-

- * The set of dml queries which are grouped logically is known as a transaction.
- * Within the transaction if any query operation execution is failed we have to cancel the transaction.
- * Within the transaction, if all queries are executed then entire transaction should have to be safe.
- * To cancel the transaction we use rollback method of connection object, to save the transaction we call the commit method of connection object

Ex:- import cx_Oracle

```
class AccountNoDoesNotExist(Exception):
```

```
    pass
```

```
con = None
```

```
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
```

```
cursor = con.cursor
```

```
try:
```

```
    sacno = int(input("enter sacno"))
```

SRI SAI BHAVANI XEROX
H.No: 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre
Ameerpet, Hyderabad-500 018

```
(input("enter transfer amt"))
```

```
query1 = "update cust set cbal = cbal - "+amt+" where sacno<="
```

```
cursor.execute(query1)
```

```
if cursor.rowcount != 1:
```

```
    print("sacno does not exist")
```

```

raise AccountNoDoesNotExist
dacno = str(input("enter dacno"))
query2 = "update cust set cbal=cbal+"+stramt+" where cauno=%s"
cursor.execute(query2)
if cursor.rowcount != 1:
    print("dacno does not exist")
    raise AccountNoDoesNotExist
cursor.close()
con.commit()

except:
    con.rollback()
    print("transaction failed")

finally:
    if con != None:
        con.close()

```

SRI SAI BHAVANI XEROX
 H.No. 7-1-209/1, Ground Floor,
 Ram Mirra Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

To create table & insert data into table

SQL> create table cust (cauno(4), cname varchar2(10), cbal number(8,3));

SQL> insert into cust values (1001, 'rama', 5000.00)

1 row created

SQL> insert into cust values (1002, 'sita', 1000.00)

1 row created

SQL> commit

Commit complete

O/P:-

enter cauno 1001
 enter transfer amt 2000
 enter dacno 1002
 >>>

SRI SAI BHAVANI XEROX
 H.No. 7-1-209/1, Ground Floor,
 Ram Mirra Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

(or) enter scno 1002
enter transfer amt 1000.00
enter dacno 1003
dacno does not exist
transaction failed
>>>

* Whenever we sent the same sql query for multiple times by changing the data by calling execute method then internally two operations will takes place at database with respect to every time calling of execute method.

- 1) query compilation
- 2) query execution.

Ex:- Import CX_Oracle

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
cursor = con.cursor()
cursor.execute("insert into cust values (1003, 'scott', 2000.00)")
cursor.execute("insert into cust values (1004, 'blake', 4000.00)")
cursor.close()
con.commit()
con.close()
```

* Even though we send the same sql query to the database for multiple times by calling execute method every time query compilation will be performed in the database so that performance of the application will be degraded.

* To overcome the above problem we use prepare method of cursor object.

* We can send the sql queries with bind variables to the database by calling prepare method of cursor object.

* Whenever query forwarded to the database through the prepare method in the database that query will be compiled & compiled query will be stored in database.

- * Whenever values are submitted to the bind variables by calling execute method of cursor object then query will be executed in the database.
- * We can submit values to the bind variables for 'n' no. of times.
- * With respect to every time submission of values to the bind variables only query execution will be takes place in database.

Ex:- import cx_Oracle

```
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
```

```
cur = con.cursor()
```

```
cur.prepare('select empno, ename, sal, deptno from emp where  
deptno = :id')
```

```
cur.execute(None, {'id': 10})
```

```
res = cur.fetchall()
```

```
for rec in res:
```

```
    print(rec)
```

```
cur.execute(None, {'id': 20})
```

```
res1 = cur.fetchall()
```

```
for rec1 in res1:
```

```
    print(rec1)
```

```
cur.close()
```

```
con.close()
```

O/p:- (7782, 'CLARK', 2450.0, 10)

(7839, 'KING', 5000.0, 10)

(7934, 'MILLER', 1300.0, 10)

(7369, 'SMITH', 800.0, 20)

(7566, 'JONES', 2975.0, 20)

(7788, 'SCOTT', 3000.0, 20)

(7876, 'ADAMS', 1100.0, 20)

(7902, 'FORD', 3000.0, 20)

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Inserting Multiple Records into one database:-

Import cx-Oracle

```
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
```

```
rows = [(1, "First"),
         (2, "Second"),
         (3, "Third"),
         (4, "Fourth"),
         (5, "Fifth"),
         (6, "Sixth"),
         (7, "Seventh")]
```

```
cur = con.cursor()
```

```
cur.bindarraysize = 7
```

```
cur.set_input_sizes(int, 10)
```

```
cur.execute many("insert into mytab(id, data) values(:1, :2)", rows)
```

```
con.commit()
```

```
cur.close()
```

Now query the results back

```
cur2 = con.cursor
```

```
cur2.execute('select * from mytab')
```

```
res = cur2.fetchall()
```

```
for rec in res:
```

```
    print(rec)
```

```
cur2.close()
```

```
con.close()
```

O/p:-

```
(1, 'First')
(2, 'Second')
(3, 'Third')
(4, 'Fourth')
(5, 'Fifth')
(6, 'Sixth')
(7, 'Seventh')
```

```
>>>
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Create table mytab

```
Create table mytab (id number(1),
                     varchar 2(10));
```

Creation of a table for bigtab:-

SQL > create table bigtab (mycol varchar2(20));

Table created.

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * Cursor array size indicates the total no. of records that can fetch from the database table at a time into the cursor object.
- * If cursor array size is small to read huge no. of records from the database table will takes longer time because no. of roundtrips between python program to the database are increased.
- * To overcome the above problem we can increase the cursor array by using cursor.arraysize = value
- * If cursor array size is high the no. of roundtrips b/w python program to the database will be decreased so that performance of the application will be increased.
- * The default cursor arraysize is cx-Oracle module is 50.

SQL > create table bigtab (mycol varchar2(20));

Table created

Plsql program:-

begin

for i in 1...10000

loop

insert into bigtab (mycol) values

(dbms_random.string ('A', 20));

end loop;

end;

/

Python program:-

import cx_Oracle

import time

con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')

start = time.time()

```

cur = con.cursor()
# cur.arraysize = 200
cur.execute('select * from bigdata')
res = cur.fetchall()
#print(res)
elapsed = (time.time() - start)
print(elapsed, "seconds")
cur.close()
con.close()

```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Executing plsql programs through python programs.

- * Instead of fetching entire data from the database table into the python program already processed data fetching into the python program will give better performance
- * To process the data within the oracle database we use plsql procedures or plsql functions.

PLSQL procedures:-

- * Procedure is a named plsql block, is used to represent business logic and procedure is stored in database permanently.
- * Procedure will be executed whenever we call the procedure.

Syntax:-

1. Create or replace procedure procedurename (var1 mode of parameter datatype, var2 mode of parameter datatype, ...)
2. is/as
3. variable declaration
4. begin
5.
6.
7. endprocedurename;
8. /.

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex1:- Procedure creation program:-

Create or replace procedure P1(a in number, b out number)
is
begin
b: a*a;
end P1;
1

* Procedure can be called ~~through~~ python program by using call.proc method of cursor object.

* Before going to call the call.proc method define the variables for the out mode parameters of procedures.

* At the time of calling the call.proc method we have to pass the values to the in mode parameters and variables to the out mode parameters.

* Database related variables can be created by using var method of cursor object.

program:-

```
import cx_Oracle
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
cur = con.cursor()
myvar = cur.var(cx_Oracle.NUMBER)
cur.callproc('P1', (10, myvar))
print(myvar.getvalue())
cur.close()
con.close()
```

O/P:- 100.0
 >>

Ex2:- procedure creation program

Create or replace procedure P2(id in number, data in varchar 2, ont out number)

is
begin
insert into mytab values (id, data);

```
commit;
select count(*) into cnt from mytab;
end P2;
/
```

Python program:-

```
import cx_Oracle
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
cur = con.cursor()
myvar = cur.Var(cx_Oracle.NUMBER)
cur.callproc('P2', (8, 'python', myvar))
print("total no of records are:", myvar.getvalue())
cur.close()
con.close()
```

O/P:- Total no of records are : 8

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Executing plsql functions:-

- * Plsql function is a named plsql prompt by using which we can represent the business logic.
- * Function should return the value
- * plsql function will be executed whenever we call that function
- * We can execute the plsql functions through python programs by calling callfunc function of cursor object.

Syntax:- Create or replace function functionname (variable1

mode of parameter datatype , variable2 mode of parameter
datatype, ---)

return datatype

is/as

variables declaration

begin

return value;

end functionname;

/

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rani Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- function creation

Create or replace function f1(a, in number)

return number

is

begin

return a*a;

end f1;

/

Python program:-

```
import cx_Oracle
```

```
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
```

```
cur = con.cursor()
```

```
res = cur.callfunc('f1', cx_Oracle.NUMBER, (9, ))
```

```
print(res)
```

```
cur.close()
```

```
con.close()
```

O/P:- 81.0

>>>

Ex:- function creation

Create or replace function f2 (tsal out number)

return number

is

cnt number(5);

begin

select count(*) into cnt from emp;

select sum(sal) into tsal from emp;

return cnt;

end f2;

/

Python program:-

```
import cx_Oracle
```

```
con = cx_Oracle.connect('scott', 'tiger', 'localhost:1521/orcl')
```

```
cur = con.cursor()
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ramt Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
myvar = cur.var(cx_Oracle.NUMBER)
res = cur.callfunc('f2', cx_Oracle.NUMBER, (myvar, ))
print("total no of employees are:", res)
print("all employees sum of sal is:", myvar.getvalue())
cur.close()
con.close()
```

op:- ('total no of employees are:', 14.0)
('all employees sum of sal is:', 29025.0)

>>>

Mysqldb & python connection:-

```
Create table mytab (id int(10), varchar(10));
insert into mytab values (1, 'django')
insert into mytab values (2, 'flask')
commit
```

python program:-

```
import MySQLdb
db = MySQLdb.connect(host="jokesh.mysql.pythonanywhere-services.com",
                      user="jokesh", passwd="sathya1234", db="jokesh$sathya")
print("connection established")
cur = db.cursor()
print(cur)
cur.execute("select * from mytab")
for p in cur.fetchall():
    print(p)
cur.close()
db.close()
print("connection closed")
```

op:- connection established
<MySQLdb.cursors.cursor object at 0x7fffa00ed2be0>
(1, 'django')
(2, 'flask')
connection closed

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Multithreading:-

- * Thread is a functionality or logic which can execute simultaneously along with the other part of the program.
- * Thread is a light weight process.
- * Any program which is under execution is known as a process.
- * We can define the threads in python by overwriting run method of thread class.
- * Thread class is a predefined class which is defined in threading module.
- * Thread in module is a predefined module.
- * If we call the run method directly the logic of the run method will be executed as a normal method logic.
- * In order to execute the logic of the run method as a we use start method of thread class.

Ex:-

```
import threading
class X(threading.Thread):
    def run(self):
        for p in range(1,101):
            print(p)
class Y(threading.Thread):
    def run(self):
        for q in range(101,201):
            print(q)
```

```
x1=X()
y1=Y()
x1.start()
y1.start()
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- import threading

class X (threading.Thread) :

def run(self):

for p in range (1,101)

print(p)

X1 = X()

X1.start()

X2 = X()

X2.start()

Ex3:- import threading

class X (threading.Thread) :

def run(self):

myfunc()

class Y (threading.Thread) :

def run(self):

myfunc()

def myfunc():

for p in range (1,101):

print(p)

X1 = X()

Y1 = Y()

X1.start()

Y1.start()

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* Whenever we run the python program then by default python interpreter creates main thread.

* After creating the main thread program execution starts from main thread.

* In the middle of the execution of the main thread if any other threads are created those threads and main thread execution will takes place one with the other.

Ex:- import threading.

class X(threading.Thread):

def run(self):

for p in range(1, 10):

class Y(threading.Thread):

print(p)

def run(self):

for q in range(101, 201):

X1 = X()

print(q)

Y1 = Y()

X1.start()

Y1.start()

for r in range(201, 301):

print(r)

Scheduling:-

- * Among multiple threads which thread as to start the execution first, how much time the thread as to execute after allocated time is over, which thread as to continue the execution.
next this comes under scheduling.

- * Scheduling is highly dynamic.

Suspending the execution of the threads in the middle of its execution temporarily

- * We can suspend the execution of the thread temporarily in the middle of its execution by calling sleep function or join method.
- * Sleep function is a predefined function which is defined in time module
- * sleep function suspend the execution of the current thread until specified time is over.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:-

```

import threading
import time
class X (threading.Thread):
    def run(self):
        for p in range(1, 101):
            print(p)
            if p == 50:
                time.sleep(10)
class Y (threading.Thread):
    def run(self):
        for q in range(101, 201):
            print(q)

```

X1=X()

Y1=Y()

X1.start()

Y1.start()

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:-

```

import threading
import time

```

class X (threading.Thread):

def run(self):

myfunc()

class Y (threading.Thread):

def run(self):

myfunc()

def myfunc():

for p in range(1, 101):

print(p)

if p == 50:

time.sleep(10)

X1=X()

Y1=Y()

X1.start()

Y1.start()

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Infinite loop with time

Ex:- import time

x = 1

while True:

 print(x)

 x = x + 1

 time.sleep(1)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,

Ram Mirra Towers, Near Satyam Theatre,

Ameerpet, Hyderabad-500 016

* Join is a predefined method, which is defined in thread class.

* Join () method suspend the execution of the current thread until specified thread execution is over.

Ex:- import threading

import time

class X (threading.Thread):

 def run(self):

 time.sleep(10)

 self.sum = 0

 for p in range(1, 101):

 self.sum = self.sum + p

class Y (threading.Thread):

 def __init__(self, x1):

 super().__init__()

 self.x1 = x1

 def run(self):

 for q in range(101, 201):

 print(q)

 if q == 150:

 self.x1.join()

 print(self.x1.sum)

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,

Ram Mirra Towers, Near Satyam Theatre,

Ameerpet, Hyderabad-500 016

x1 = X()

y1 = Y(x1)

x1.start()

y1.start()

```

Ex:- import threading
      import time
      class X(threading.Thread):
          def run(self):
              time.sleep(10)
              for p in range(1,101):
                  print(p)

```

```

class Y(threading.Thread):
    def run(self):
        for q in range(101,201):
            print(q)

```

```

x1 = X()
y1 = Y()
x1.start()
y1.start()
lst = [x1,y1]

```

```

for r in range(201,301):
    print(r)
for a in lst:
    a.join()

```

Synchronization:-

- * The concept of avoiding multiple threads to access the same logic at a time is called as synchronization.
- * Synchronization can be implemented by calling acquire & release method of lock class.
- * Lock class is present in threading module
- * Synchronization degrades the performance of the application
- * If any specific business requirements need synchronization then only implement synchronization.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:- import threading
import time
class X (threading.Thread):
def run(self):

lc.acquire()

my func ("django")

lc.release()

class Y (threading.Thread):

def run(self):

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,

Ram Mira Towers, Near Satyam Theatre,

Ameerpet, Hyderabad-500 016

lc.acquire()

my func ("flask")

lc.release()

def myfunc(msg):

print("hello [", msg)

time.sleep(10)

print("] world]")

lc = threading.Lock()

x1 = X()

y1 = Y()

x1.start()

y1.start()

a = [x1, y1]

for b in a:

b.join()

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,

Ram Mira Towers, Near Satyam Theatre,

Ameerpet, Hyderabad-500 016

O/P:- [hello [django

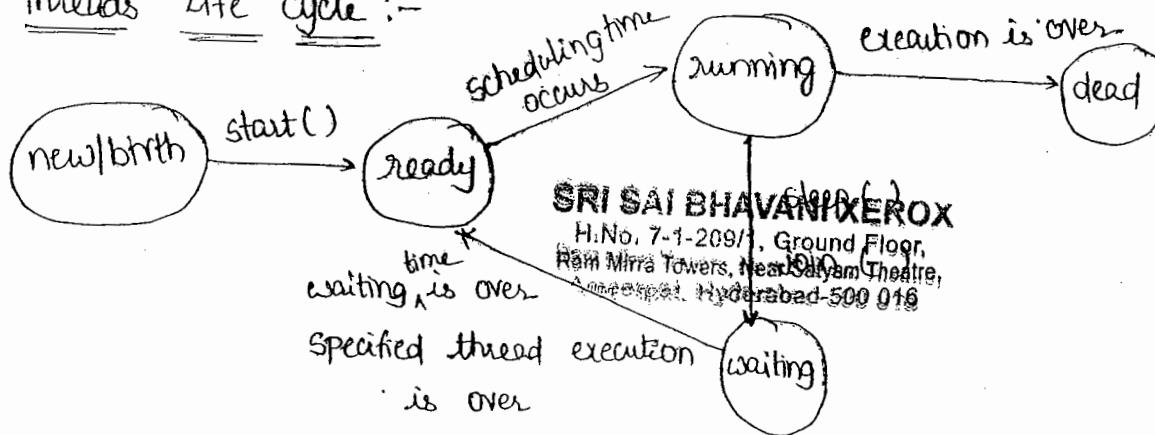
] world]

[hello [flask

] world]

>>>

Threads Life cycle :-



- * Creating the object of a class which is over writing run method of thread class is known as a creating thread.
- * Whenever thread is created then we call thread is in new state or birth state thread.
- * Whenever we call the start method on the new state threads then those threads will be forwarded for scheduling.
- * The threads which are forwarded for scheduling are known as ready state threads
- * Whenever scheduling time occurs, ready state thread starts execution.
- * The threads which are executing are known as running state threads.
- * Whenever sleep, join methods are called on the running state threads then immediately those threads will wait.
- * The threads which are waiting are known as waiting state threads
- * Whenever waiting time is over or specified thread execution is over then immediately waiting state threads are forwarded for scheduling
- * If running state threads execution is over then immediately those threads execution will be terminated.
- * The threads which execution is terminated are known as dead state threads.

Iterators:-

- * Iterator is a protocol, if any class is implementing iterator protocol, that class objects we can convert in the form of iterable objects.
- * If any class contains two special methods called `__iter__` and `__next__` the we call that class is implementing or satisfying iterator protocol.
- * We can create the objects for the iterator protocol implemented classes and we can convert those objects in the form of iterable objects by calling `iter` function on those objects.
- * Whenever we call the `iter` function on any object then internally `__iter__` method of that object represented class will be executed.
- * Iterable object can store the zero or more elements.
- * To get the one by one element from the iterable objects we call the `next` function on iterable objects.
- * Whenever we call the `next` function on iterable object then internally that object represented class `__next__` method will be executed.
- * According to the iterator protocol rules `__iter__` method should return the object and `__next__` method should contain the stop iteration exception rising statement.

Ex:- class PowTwo:

```
def __init__(self, max=0):  
    self.max = max  
    self.n = 0  
def __iter__(self):  
    return self  
def __next__(self):  
    if self.n <= self.max:  
        result = 2 ** self.n  
        self.n += 1  
        return result
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

else:

raise StopIteration

P1 = PowTwo(5)

it1 = iter(P1)

print(next(it1))

print(next(it1))

~~print(next(it1))~~

~~print(next(it1))~~

~~print(next(it1))~~

~~print(next(it1))~~

~~print(next(it1))~~

O/P:
1
2
4
6
8
16
32

raise StopIteration

Ex:- class PowTwo:

def __init__(self, max=0):

self.max = max

def __iter__(self):

self.n = 0

return self

def __next__(self):

if self.n <= self.max:

result = 2 ** self.n

self.n += 1

return result

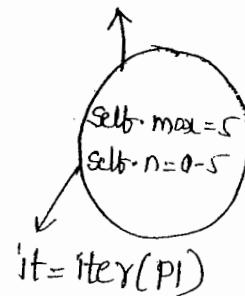
else:

raise StopIteration

P1 = PowTwo(5)

it1 = iter(P1)

P1 = PowTwo(5)



SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Rammurthy Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

while True :

try :

 print (next(it)):

except (StopIteration) :

 break

Output:

1

2
4

8

16

32

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,

Ram Mirra Towers, Near Satyam Theatre,

Ameerpet, Hyderabad-500 016

* for loop is implemented in python language as follows

for element in iterable:

 iter_obj = iter(iterable)

 while True:

 try:

 element = next(iter_obj)

 except (StopIteration):

 break

* for loop takes the given object, convert that object in the form of iterable object & gets the one by one element from the iterable object.

* while getting the one by value element from the iterable object if stop iteration exception is raised then for loop internally handle that exception.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:

```
class PowTwo:  
    def __init__(self, max=0):  
        self.max = max  
    def __iter__(self):  
        self.n = 0  
        return self  
    def __next__(self):  
        if self.n <= self.max:  
            result = 2 ** self.n  
            self.n += 1  
            return result  
        else:  
            raise StopIteration  
for x in PowTwo(5):  
    print(x)
```

Output:

1

2

4

8

16

32

>>>

Ex2:

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

```
it = iter(x)
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
print(next(it))
```

```
print(next(it))
```

```
print(next(it))
```

Point SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

```
print(next(it))
```

Q.P.:

10

20

30

40

stop iteration exception is raised

>>>

Eg:

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

```
it = iter(x)
```

```
while True:
```

```
    try:
```

```
        print(next(it))
```

```
    except (StopIteration):
```

```
        break
```

Q.P.:

10

20

30

40

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:

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

```
for p in x:
```

```
    point(p)
```

O/P:

```
10
```

```
20
```

```
30
```

```
40
```

>>>

Ex:

```
class X:
```

```
    def m1(self):
```

```
        point ("in m1 of X")
```

```
    for p in X():
```

```
        point(p)
```

O/P:

Type error

Ex:

```
class Inffilter:
```

" " "

Infinite iterator to return all odd numbers

```
def __iter__(self):
```

```
    self.num = 1
```

```
    return self
```

```
def __next__(self):
```

```
    num = self.num
```

```
    self.num += 2
```

```
    return num
```

SRI SAI BHAVANI XEROX
 H.No. 7-1-209/1, Ground Floor,
 Ram Mirra Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
 H.No. 7-1-209/1, Ground Floor,
 Ram Mirra Towers, Near Satyam Theatre,
 Ameerpet, Hyderabad-500 016

```
i=Infilter()
```

```
it=iter(i)
```

```
while True:
```

```
    try:
```

```
        print(next(it))
```

```
    except (StopIteration):
```

```
        break
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

on sairam

Generators :

- * Generator is a function, which contains one or more yield statements.
- * whenever we call the generator function without executing the function logic it returns iterable object i.e., generator function internally implementing iterator protocol.
- * whenever we call the next function on the iterable object which is given by the generator function then that generator function logic execution will be started and control comes out from the generator function execution if control reached to yield statement.
- * Once again if we call the next function on the iterable object which is given by the generator function then generator function logic execution starts from previous yield statement to the next yield statement.
- * if no more yield statements in the generator function it raises stop iteration exception.
- * whenever control reached to the yield statement of a function without terminating the function execution pause the function execution & control comes out from the function execution.

E:

```
def my_gen():
```

```
    n = 1
```

```
    print('This is printed first')
```

```
    yield n
```

```
    n += 1
```

```
    print('This is printed second')
```

```
    yield n
```

```
    n += 1
```

```
    print('This is printed at last')
```

```
    yield n
```

```
it = my_gen()
```

```
print(it)
```

```
print(next(it))
```

```
print(next(it))
```

```
print(next(it))
```

```
print(next(it))
```

O/P.

```
<generator object my_gen at 0XD24B71B0>
```

```
This is printed first
```

```
1
```

```
This is printed second
```

```
2
```

```
This is printed last
```

```
3
```

```
'Stop iteration exception raised'
```

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Amarpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Amarpet, Hyderabad-500 016

E2:
def rev-str(my-str):

 length = len(my-str)

 for i in range(length - 1, -1, -1):

 yield my-str[i]

it = rev-str("lokesh")

print(next(it))

print(next(it))

print(next(it))

print(next(it))

print(next(it))

print(next(it))

print(next(it))

O/p:

n ↳ py... --

h

S

e

K

o

l

"stop iteration exception"

E1:

def rev-str(my-str):

 length = len(my-str)

 for i in range(length - 1, -1, -1):

 yield my-str[i]

it = rev-str("lokesh")

while True:

 try:

 print(next(it))

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

except (stop iteration):

- break

Ex:

n7.py

h

s

c

k

o

l

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex:

```
def rev_str(my_str):
```

```
    length = len(my_str)
```

```
    for i in range(length - 1, -1, -1):
```

```
        yield my_str[i]
```

```
for p in rev_str("lokesh"):
```

```
    print(p)
```

Output:

n7.py

h

s

e

k

o

>>>

SRI SAI BHAVANI XEROX

H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Ex: By using tuple comprehension

```
x = (p for p in range(5))
```

```
^print(x)
```

```
for i in x:
```

```
    print(i)
```

Infinite Generator

E1:

```
def all_even():
```

```
    n=0
```

```
    while True:
```

```
        yield n
```

```
        n+=2
```

```
-for p in all_even():
```

```
    print(p)
```

E2:

```
def PowTwoGen(max=0):
```

```
    n=0
```

```
    while n<=max:
```

```
        yield 2**n
```

```
        n+=1
```

```
-for a in PowTwoGen(5):
```

```
    print(a)
```

O/p:

n10.py

1

2

4

8

16

32

>>

* Generator implementation is simple when compared to the

iteratur implementation.

* Generators are memory efficient.

* By using Generators we can represent the infinite strings.

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Closures

Nested Functions:

- * A function which is ~~SRI SAI BHAVANI XEROX~~ H.No. 7-1-209/1, Ground Floor, Ram Mirra Towers, Near Satyam Theatre, Ameerpet, Hyderabad-500 016 contained in another function is known as a nested function or inner function.
- * A function which contains another function is known as a enclosing function or outer function
- * whenever we call the outer function by default outer function only will be executed.
- * whenever we call the inner function then only inner function will be executed.
- * whenever we call inner function cannot be called directly outside of the outer function.
- * inner function can be called directly outside or within the outer function
- * if we call the inner function within the outer function then that inner function will be executed whenever we call outer function

Ex:

```
def outer():
    print ("in outer")
    def inner():
        print ("in inner")
    inner()
outer()
```

o/p:

st.py

in inner

in outer

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

* We can call the inner function of enclosing function, outside of that enclosing function by returning inner function object through the outer function

Eg:

```
def outer():
    print("in outer")
    def inner():
        print("in inner")
    return inner
myfunc = outer()
myfunc()
```

O/P:

```
in outer
in inner
>>>
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

- * After returning the inner function object by the outer function we can delete the outer function
- * Even though we delete the outer function still we can access inner function

Eg:

```
def outer():
    print("in outer")
    def inner(a):
        print(a)
        print("in inner")
    return inner
myfunc = outer()
del outer
```

```
myfunc(1000)
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

O/P:

in outer

1000

in inner

error

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

closure meaning:

* Any function which is satisfying following rules and regulations
then we call that function as a closure.

1, function should contain nested function

2, The nested function must refer to a value defined in the
enclosing function.

3, The enclosing function must return the nested function.

Eg:

```
def outer(a):
```

```
    print("in outer")
```

```
    def inner():
```

```
        print(a)
```

```
        print("in inner")
```

```
    return inner
```

```
my_func=outer(1000)
```

```
my_func()
```

O/P:

in outer

1000

in inner

>>>

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mitra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

E1:

```

def outer(a):
    print("in outer")
    def inner():
        print(a)
        print("in inner")
    return inner

```

```
myfunc = outer(1000)
```

```
del outer
```

```
myfunc()
```

O/P:

```
in outer
```

```
1000
```

```
in inner
```

```
>>>
```

* closures can avoid the use of global values & provides some form of data hiding

* when there are few methods (one method in most cases) - to be implemented recommended to define closure

* Instead of defining single class with single method recommended to define closure

E1:

```
def make_multiplier(x):
    def multiplier():

```

```
        return x*n
```

```
    return multiplier
```

```
times3 = make_multiplier_of(3)
```

```
print(times3(5))
```

O/P:

```
15
```

```
>>>
```

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

Decorators:

- Decorators are used to add some functionality to the existing code.
- The function which contains the logic is known as decorator function.
- The logic which we want add to the decorator function is implemented in other function & we call that function as a decorated function.
- Decorated function object will be passed as a parameter to decorator function whenever we call the decorated function.
- Decorated function executes & returns the object.
- The return object of the decorator function is going to be executed automatically.

Ex:

```
def make_pretty(func):  
    def inner():  
        print("I got decorated")  
        func()  
        return inner
```

@ make_pretty

```
def ordinary():  
    print("I am ordinary")  
ordinary()
```

O/P:

I got decorated

I am ordinary

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

F12.

def smart_divide

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016

def inner(a, b):

print("I am going to divide", a, "and", b)

if b == 0

print("whoops! cannot divide")

return None

func(a, b)

return inner

@ smart_divide

def divide(a, b):

print(a, b)

divide(10, 5).

o/p:

I am going to divide 10, and 5

2.0

SRI SAI BHAVANI XEROX
H.No. 7-1-209/1, Ground Floor,
Ram Mirra Towers, Near Satyam Theatre,
Ameerpet, Hyderabad-500 016