PYTHON

BY

Mr. LOKESH sir

LOKESH IT

Flat: 302, Sree Swati Anukar, Beside: Aditya Trade Centre,
Ameerpet, Hyderabad - 16.
Ph: + 91 90307 49297, +91 80740 96308

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	✓	✓
* Gaming	✓	✓
* Animation	X	X

Addition of 2 no's:-

Program:-

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

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

IOT - Internet of things.

① Automation

* They are using python in Robotics.
* Robotics are electronic device. Generally robotics don't have intelligence. By Applying Artificial intelligence alogrithms. 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.

  After reading the data, the data is represented in pie-chart, Bargraph, 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

Java → STRUTS, SPRING, JSF

.NET → MVC

PYTHON → DJANGO, FLASK, WEB2PY

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 N/w with IOT:-

* IOT allows you to connect to various n/w's. It can also be connected to the smartphones, washing machines etc...
* 90% 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.

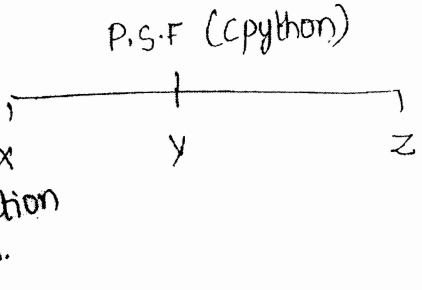
* Python supports the four, procedure, object, scripting and module programming language
* 'C' is suitable for gaming and hardware drivers.

* python was implemented by Gyro vendrosom 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 of python software foundation has taken as a third party of python by Gyro vendrosom
* Java author was sunmicro 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 text version of python software available (old)
* 3.6 text version of python software available (new)
* Python is platform independent.
* Gyro vendrosom 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

5) Web scrapping techniques
6) N/w with IOT applications
7) Administration script
8) Test cases
9) GUI applications
10) Gaming applications
11) Animation applications. . . .

* 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



* 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 installer file from the WWW.python.org.com website.

Installation :-

* Click on the downloaded executable installer
* Click on 'Run'
* Click on costmize 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 35-32;

↓

Click on 'ok'

↓

Click on 'ok'

↓

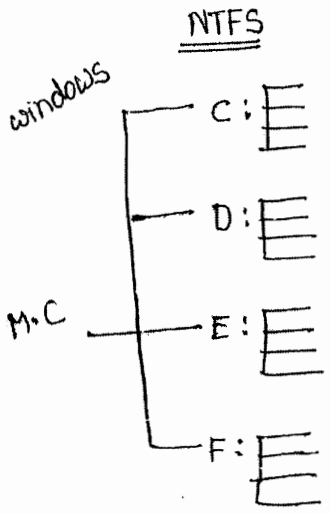
Click on 'ok'

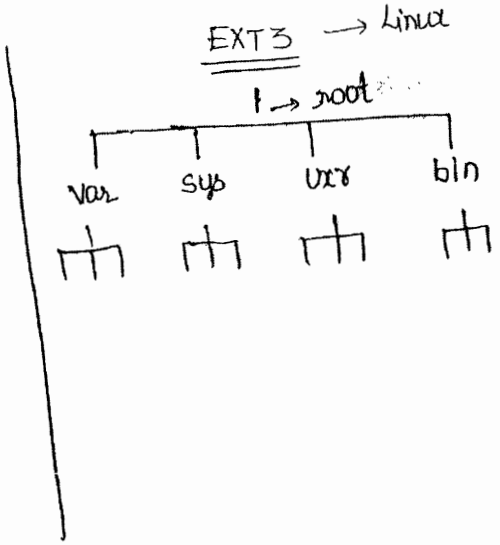
Now path setting is secured.

* python programs can be developed in two modes.

1) Interactive mode

2) Batch mode.





* Windows following file format NTFS files 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 explicity)
2) Batch mode (writting 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.

Interactive mode:-

* The concept of submitting one by one python statements explicity in the python interpreter is known as "interactive mode"
* Inorder to submit the one by one python statements explicity to the python interpretor 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.

never we exit from python command line shell environment. the world 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 learning python and to test predefined functions and functionality

Batch Mode:-

In the concept of writing the group of pythons statements in a file, save the file with extension .py and submit that entire file to the python interpretor 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 9am python folder with the demo.py.
* Open command prompt and execute following commands.

```
.python D:\9am python\Demo.py

3000
-1000
2000.000
```

* Save another method if we correctly set the path

D:

D:|>d gam python

D:|iam python > python Demo.py

3000
-1000
2000.000

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 ....

wnload 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

* Righ click on qamsample proj
  ↓
  Click on 'new'
  ↓
  Click on python file
  ↓
  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'

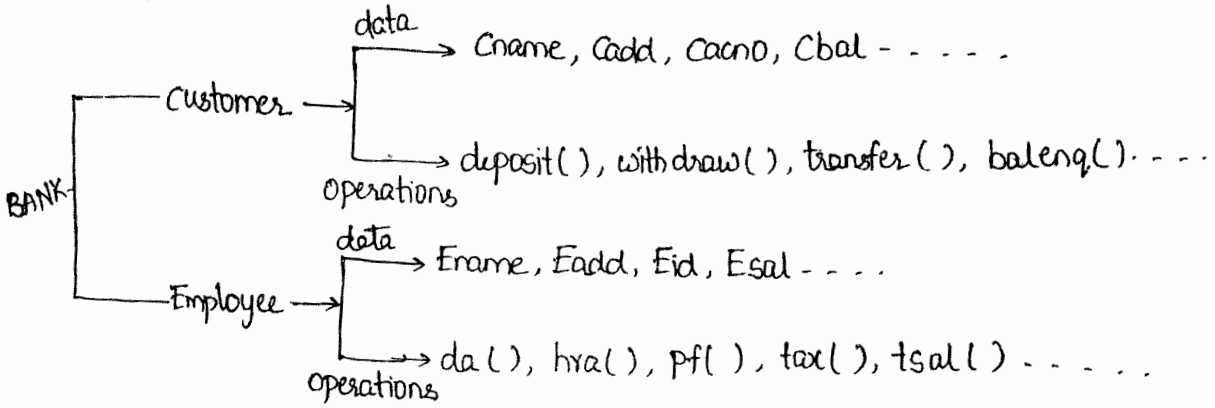
→ Click on 'Run'

→ Click on 'Demo'

o/p:-
```
300
-100
20,000
```

Business Applications:-

* Any business organisation use case or scenerio contains two parts they are
  1) data
  2) operations.



* Data of organisation use cases 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.

...mming languages supports two different varieties of data ...es:

1) Static data types
2) Dynamic data types

Static Data types:-

In static data types supported languages programmer should define the data type to the variable explicity

* 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 explicity
* 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 on 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))
print(id(a))
b=123.123
print(b)
print(type(b))
print(id(b))
c=3+4j
print(c)
print(type(c))
print(id(c))
```

type(1)
print(1)
id(1)
↓
address

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

o/p:-

```
1000
<class 'int'>
36785936
123.123
<class 'float'>
36786160
(3+4j)
<class 'complex'>
36654496
True
<class 'bool'>
1809368336
sathya
<class 'str'>
38668064
```

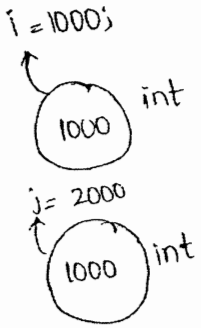
* Python supports two types of objects. They are

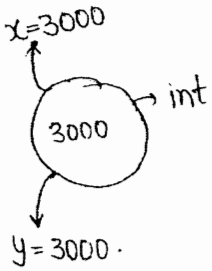
1) Immutable objects
2) Mutable objects.

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 interpretor verifies is already any object is available. in memory location with same content or not.
* If already object is not available then python interpretor 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.





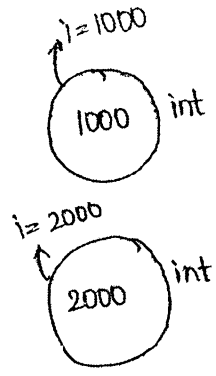
Program:-

```
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))
```

O/P:-

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



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
>>>
```

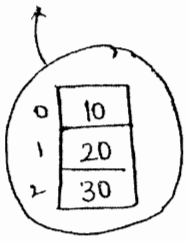
* Int, float, complex, bool, str, tuple are immutable objects.
* Immutable objects performance is high
* Applying Iterations on immutable objects takes less time.
* All fundamental types represented classes objects and tuple class objects are immutable objects.

Mutable objects:-

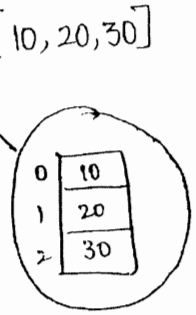
The objects which allows to modify the contents of those objects are known as 'Mutable objects'.

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

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))
```

O/P:- [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)
```

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

Op:- [10, 20, 30]
<class 'list'>
37565624
[10, 20, 30]
<class 'list'>
37565624

* list, set, dict classes objects are mutable objects
* Mutable objects perbormance 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 " "
  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 indexes we can read the data from the str objects.

Program:-

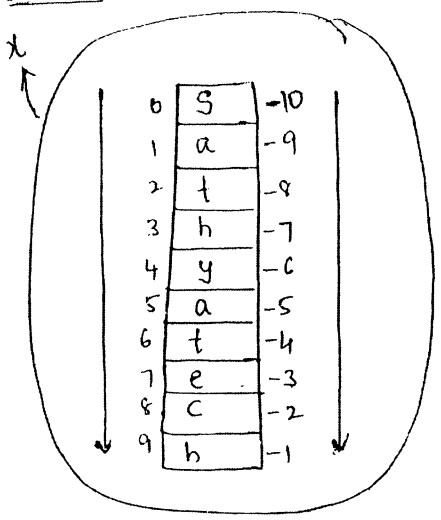
```
x='sathya technologies'
print(x)
print(type(x))
print(id(x))
y="""ameerpet
hyderabad
telangagana"""
print(y)
print(type(y))
print(id(y))
```

output:-

```
sathya technologies
<class 'str'>
6099792
ameerpet
hyderabad
telangana
<class 'str'>
6080456
>>>
```

Index & Slicing method

Ex:-



Program:-

```
x = 'sathyatech'
print (x)
print (x[2])
print (x[-2])
print (x[1:5])        → exclusive
                      ↓ inclusive
print (x[5:-1])
print (x[ :7])
print (x[4:1])
print (x[-6:-1])
print (x[-2:-8])      - ① — no
print (x[2:-2])       - ⑩ thyati
print (x[8:-7])       - ⑪ — no
```

output:-

sathyatech
t
c
athy
atech
sathyat
yatec
thyate
>>>

str
list
tuple
set
dict
} Iterable objects

* Iterable objects has predefined function i.e., len()
* len() built in function is used to count the number of characters.

Eg:- x = 'sathyatech'
print(x)
print(len(x))
y = 1000
print(y)
print(len(y))

o/p:- sathyatech
10
1000

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

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
>>>
```

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
 >>>
```

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))
```

o/p:- 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"
print (a+b)
print (c+d)
print (a+e)
print (a+str(c)) → We does not [illegible]
print (c+int(e)) → Here we can modify either 'c' → string (or) e → int
print (str(c)+e)   both are possible
```

o/p:- sathyatech
3000
Sathya1234
Sathya1000
2234
10001234
>>>

program:-

```
a="sathya"
print (id (a))
print (a)
print (len(a))
#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) comparision (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, substraction, 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 = 50625 (y^15)
```

Comparision Operators:-

Comparision operators are used to compare the data of the objects which are pointed by the operands.

Ex:-
```
x=10
y=12
print('x>y is ', x>y)
print('x<y is ', x<y)
print('x==y is ', x==y)
print('x!=y is ', x!=y)
print('x>=y is ', x>=y)
print('x<=y is ', x<=y)
```

O/P:-
```
x>y is false
x<y is true
x==y is false
x!=y is true
x>=y is false
x<=y is true
>>>
```

Logical Operators:-

logical operators are used to perform the mathematical logical operations

Ex:-
```
x= True
y= False
print('x and y is', x and y)
print('x or y is', x or y)
print('not x is', not x)
```

o/p:- 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

o/p:- 0
2.5
14
-11
2
40
>>>

Assignment operators:-

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

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.

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
```

```
3000
37523824
3000
37523824
True
False
[10, 20, 30]
 37125176
[10, 20, 30]
37125576
False
True
>>>
```

2) Membership Operator:-

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

program:-
```
i='sathya'
print('t' in i)
print('t' not in i)
print('p' in i)
print('p' not in i)
```

o/p:-
```
True
False
False
True
>>>
```

program:-
```
i=1000
print('4' in i)
```

o/p:-
```
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

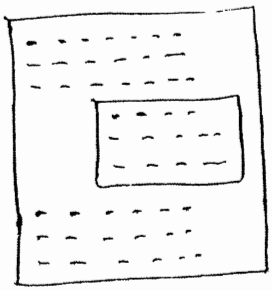
Condition:-

* 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

1) 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
>>>
```

```
begin
enter positive no45
end
>>>
```

2) else :-

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

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

Syntax :-

```
if condition :-
        statement 1
        statement 2
        - - - - - -
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")
```

o/p:- begin
enter positive no25
given no is >=2 digit no
end
>>>
begin
enter positive no5
given no is 1 digit no
end.

3) elif:-

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

Syntax:-

```
if condition:
    statement 1
    statement 2
    - - - - -
Elif condition:
    statement 1
    statement 2
    - - - - -
```

* Elif block preceeding 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")
    elif i<1000:
        print("given no is 3 digit no")
    else:
        print("given no is >=4 digit no")
    print("end")
```

o/p:-

```
begin
enter positive no45
given no is 2 digit no
end
>>>
begin
enter positive no 123
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
    .....
```

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
>>>
```

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
in while else
end
>>>
```

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")
```

```
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")
```

o/p:-
```
begin
Welcome
Welcome
Welcome
end
>>>
```

Continue:- (skipping the statements)

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

* Whenever control reach 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? aaa
enter username sathya
Hello, sathya. what is the password? lokesh
Access granted
```

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 = range(10)
for p in a:
    print(p)
b = range(10,20)
for q in b:
    print(q)
c = range(20,30,2)
for r in c:
    print(r)
d = range(40,30,-1)
for x in d:
    print(x)
```

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
>>>

* 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 = 'sathya technologies'
print(x)
print (len(x))
print (x.capitalize())
print (x.title())
print (x.upper())
print (x.lower())
print (x.isdigit())
y = '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
* Incession 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))
```

```
c = [10, 20, 30, 40, 50]   → Homogeneous.
print (c)
d = [10, 12.12, True, 3+4j, '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)
```

(1) 30 - 4
(2) 40 - 3
(3) 50 - 2
(4) 60 - 1

o/p:- [10, 20, 30, 40, 50, 60]

30
50
[20, 30, 40]
[ ]
[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
>>>

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>

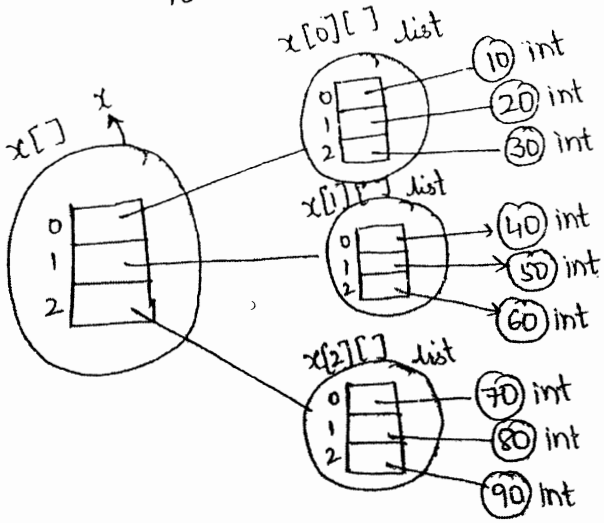
10
20
30

[40, 50, 60] class <list>

40
50
60

[70, 80, 90] class <list>

70
80
90



Example:-

```
x=[10,20,30,40,50]
i=int(input("Enter search element"))
if i in x:
    print("element is found")
else:
    print("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,True,123.123]
print(x,type(x))
a,b,c=x
print(a,type(a))
print(b,type(b))
print(c,type(c))
```

O/P:- [100, True, 123.123] <class 'list'>

100 <class 'int'>

True <class 'bool'>

123.123 <class 'float'>

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)
print(x)
print(x.count(10))
print(x.index(30))
y = x.copy()
print(y)
x.insert(2, 123)        # 2 → index
print(x)
x.remove(30)
print(x)
x.pop(3)
```

```
print(x)
z= [80,70,90]
print(z)
x.extend(z)
print(x)
x.sort(reverse = True)
print(x)
x.reverse()
print(x)
x.clear()
print(x)
```

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]
[10, 20, 123, 10, 50]
[80, 70, 90]
[ 10, 20, 123, 10, 50, 80, 70, 90]
[123, 90, 80, 70, 50, 20, 10, 10]
[10, 10, 20, 50, 70, 80, 90, 123]
[ ]
>>>
```

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 for p 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)
```

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

Example:-

```
x = [p*p for p in range(10)]
print(x)
y = [q*q for q in range(10,20) 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:-

```
noprimes = [j for i in range (2,8) for j in range (i*2, 50, i)]
primes = [x for x in range (2,50) if x not in noprimes]
print (primes)
```

(or)

```
noprimes = [ ]
for i in range (2,8):
        for j in range (i*2, 50, i):
                noprimes append (j)
primes = [ ]
for x in range (2,50):
        if x not in noprimes:
                primes append (x)
print (primes)
```

o/p:-

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

Working:-

noprimes = [4, 6, 8 ---- 48
6, 9, 12 ---- 48
8, 12, 16 ---- 48
10, 15, 20, ---- 45
12, 18, 24 ---- 48
14, 21, 28 ---- 49]

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
```

Example:-

```
line = 'the quick brown fox jumps over the lazy dog'
words = line.split()
stuff = [[w.upper(), w.lower(), len(w)] for 'win 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]
```

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 immutable objects
* Iteration order is preserved
* Duplicate elements are allowed
* Heterogeneous elements are allowed
* tuple supports both positive and negative indexing.
* The elements of the tuple can be mutable or immutable

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)  → Insertions duplicate
print(p)
q = (100, 123.123, True, "sathya")  → Heterogeneous
print(q)
```

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, 'sathya')
>>>

Example:-

```
X = [10,20,30,40,50,60]
print(X)
print(X[2])
print(X[-2])
print(X[2:5])
print(X[-6:-2])
```

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)
suml=0
i=0
```

0+10
=10
10+20

```
While i < len(x):
        sum1 = sum1 + x[i]
        i = i+1
print (sum1)
```

o/p:-
```
150
150
>>>
```

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
```

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))

Example:-

```
x = (10,12.12,True)
print(x)
a,b,c = x
print(a, type(a))
print(b, type(b))
print(c, type(c))
x[1] = 1000
```

→ cannot be modified because it is immutable object

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))
print (x. count (10))
print (x. count (40))
```

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

```
2
4
2
1
>>>
```

Differences b/w List and Tuple

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

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 = set()
print(y)
print(type(y))
print(len(y))
z = {10, 20, 30, 40, 50}
print(z)
P = {10, 20, 10, 20, 10, 30}
print(P)
q = {100, 12.12, True, "sathya", 3+5j}
print(q)
```

→ Duplicate elements are not allowed

→ Insertion order is not preserved

Heterogeneous elements are allowed

O/P:- {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)
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))
```

O/P:- {40, 10, 20, 50, 30}
True
40
10
20
50
30

{(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

Eg:- 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() → first no will be removed
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. symmetic - difference (B))
print (B. symmetic - difference (A))
```

o/p:- {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}
{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)
```

o/p:- [10, 20, 10, 30, 10, 20]

{10, 20, 30}

Sathya

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

>>>

Set Comprehension:-

* 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 for p in range(10)}

print(x)

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

print (y)

z = {r for r in range (20, 30) if r % 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:-

* 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.
* Hetereogeneous keys and Hetereogeneous values are allowed.

Example:-

```
x={ }
print(x)
print(type(x))
print(len(x))
y=dict
print(y)
print(type(y))
print(len(y))
z={"python":99,"java":90,"oracle":82,"django":97}
print(z)
```

O/P:-

```
{ }
<class 'dict'>
0
{ }
<class 'dict'>
0
{'oracle':82,'python':99,'java':90,'django':97}
>>>
```

Example:-

```
x={"python":99,"java":90,"oracle":82,"django":99}
print(x)
x["oracle"]=92
print(x)
y={100:"hyd",12.12:"bang",True:"pune",3+4j:"chennai"}
print(y)
z={10:1000,20:12.12,30:True,40:3+5j}
print(z)
p={(1,2,3):"flask","hyd":[5,6,7]}
print(p)
```

(keys → , values →)

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',True:'pune'}
{40:(3+5j),10:1000,20:12.12,30: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)
```

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

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)
```

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
>>>

Dictionary Comprehensions:-

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

Ex:- x={p:p*p in range(5)}
print(x)
y={q:q*q in range.(5,15) if q%2!=0}
print(y)

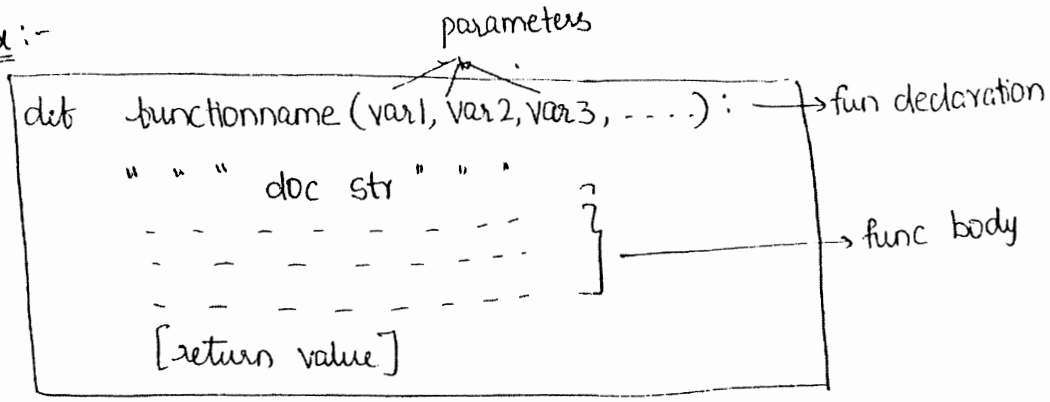
o/p:- {0:0, 1:1, 2:4, 3:9, 4:16}
{9:81, 11:121, 13:169, 5:25, 7:49}
>>>

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.

Syntax :-



Example :-
```
print ("begin")
def f1 ( ) :
    print ("in f1")
f1 ( )
f1 ( )
print ("end")
```

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

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( )
```

Op:-
```
30
300
579
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)
```

o/p:-
```
30
1020
300
579
>>>
```

Ex:- (a → non-default, b=20 → default)
```
def add(a, b=20):
    c=a+b
    print(c)

add(1000)
add(100, 200)
add(123, 456)
```

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)
```

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 support 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", "lokesh")
```

O/P:-
```
hello sathya good morning
hello good evening lokesh
>>>
```

Keyword 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
```

* After defining keyword arguments we are 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", "lokesh")
```

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)
```

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)
```

o/p:-
```
( )
1000
(10, )
20
(1,2,3,4)
5
>>>
```

Return Statement:-

* 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 = add (100, 200)
print (x)
add (10, 20)
```

O/p:-
```
300
>>>
```

Ex:-
```
def absolute (a):
    if a <0 :
        return -a
    else
        return a
    print ("end of absolute")
x = absolute (-5)
print (x)
y= absolute (15)
print (y)
```

O/p:-
```
5
15
>>>
```

* By default function returns the "none" value

Ex:-
```
def add (a,b):
    c= a+b
x = add (10, 20)
print (x)
```

O/p:-
```
None
>>>
```

* 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()
```

o/p:-
```
<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.

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.

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

o/p:-
```
2000
2000
>>>
```

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
 .
 .
 .
```

Ex:-
```
i=0
def f1 ( ):
    print ("in f1")
    global i
    i = i+1
    while i<=5
        f1 ( )
```

```
f1()
```

o/p:-
```
in f1
in f1
in f1
in f1
in f1
in f1
>>>
```

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)
```

o/p:- 100
>>>

* The above function is equivalent to the following function

Ex:-
```
def myfunc(x):
    return x * x
a = myfunc(10)
print(a)
```

o/p:- 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 values.

Modules:-

* Every python file itself is known as a module
* A module can obtain contain global variables, executable statement, functions, and classes.

* 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

1) 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")
```

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)
```

o/p:- 2000
In f2 of mod2
1000
In f1 of mod1
3.141592653589793
10.0
>>>

* 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)
```

o/p:- 2000
in f2 of mod2
1000
in f1 of 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 of mod2")
print(b)
f2()
print(a)
f1()
print(pi)
x=sqrt(100)
print(x)
```

o/p:- 2000
in f2 of mod2
1000
in f1 of 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)
```

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 interpretor 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 explicity

Example:-
```
import math
print(math.pi)
print(math.sqrt(100))
print(len("sathya"))
```

o/p:- 3.14159265
10.0
6
>>>

Re-loading a module:-

* Even though we import one module for morethan 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)
```

o/p:- hi from mod3
2000
in f2 of mod4
1000
in f1 of mod3
hi from mod3
>>>

Modules search path:-

* By default python interpretor 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 interpretor 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

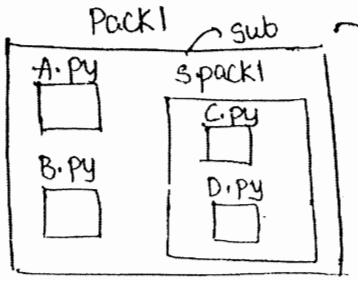
Ex:-
```
import mod3
from math import pi, sqrt
b= 2000
def f2():
    print("in f2 of mod4")
print(dir())
```

o/p:- hi from mod3

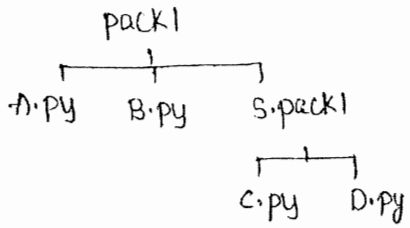
['__annotations__', '__built-ins__', '__doc__', '__file__', '__loader__', '__name__' '__package__', '__spec__', 'b', 'f2', 'mod3', 'pi', 'sqrt']

Packages:-

* Package is nothing but a folder 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.modulename
* We can import the modules of the subpackages by using packagename.subpackagename.modulename



import pack1.A
import pack1.spack1.C



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.spack1")
```

D.py

```
d = 4000
def f4():
    print("in f4 of D of pack1.spack1")
```

Test.py

```
import pack1.A
from pack1.B import b,f2
import pack1.spack1.c as p
from pack1.spack1.D import *
print(pack1.A.a)
pack1.A.f1()
print(b)
f2()
print(p.c)
p.f3()
print(d)
f4()
```

O/p:-

```
1000
in f1 of A of pack1
2000
in f2 of B of pack1
3000
in f3 of c of pack1.spack1
4000
in f4 of D of pack1.spack1
>>>
```

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

Mode	Description
'r'	- open a file for reading (default)
'w'	- open a file for writting. Creates a new file if it does not exist or truncates the file if it exist
'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 point the file 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
>>>
```

Writing the data into the file

Ex:- x = open ("myfile.txt", "w")
x.write ("sathya tech in ameerpet in")
x.close ()

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

Ex:- x = open ("myfile.txt", "a") append - add
x.write ("hyderabad in telangana")
x.close ()

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) Run time errors.

1) Syntax errors:-

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

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 interpretor varifies 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.

Abnormal termination:-

* The concept of terminating the program in the middle of its execution without executing last statement of the main module is known as a 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 rised, 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.

Try block:-

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

Syntax:- try:

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

* 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.

Ex:-
```
print ("begin")
i= input ("enter fno")
j= input ("enter sno")
x = int (i)
y= int(j)
try :
    z= x/y
    print(z)
except (zero DivisionError): → 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 fno100
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

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

o/p:-
```
begin
enter fno 100
enter sno 0
error occured
end
>>>
```

(or)

o/p:-
```
begin
enter fno 100
enter sno abc
error occured
end
>>>
```

Single try with multiple except blocks:-

* To handle that 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 rised 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.

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 (value Error) :
    print ("enter numerical values only")
except :
    print ("error occured")
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
>>>
```

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

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 occured")
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:
    - - - -
    - - - -
    - - - -
```

* Finally block should be preceded by either try block or except block.
* The set of statements which are compulsory to execute whether exception is rised or not rised even though exception

is rised 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
>>>
```

Ex:-
```
x = None
try:
    x = open("myfile.txt")
    print("file is opened")
    print(x.read())
except:
    print("error occured")
finally:
    if x != None
        x.close()
        print("file is closed")
```

O/p:-
```
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 rised 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 rised in inner try block then control goes to innertry except block
* If the inner try except block is not handle that exception then control will goes to outer try except block

Ex:-
```
try:
    print("in try 1")
    try:
        print("in try 2")
    except:
        print("in except 2")
    finally:
        print("in finally 2")
except:
    print("in except 1")
    try:
        print("in try 3")
    except:
        print("in except 3")
    finally:
        print("in finally 3")
finally:
    print("in finally 1")
    try:
        print("in try 4")
    except:
        print("in except 4")
    finally:
        print("in finally 4")
```

O/P:-
```
in try 1
in try 2
in finally 2
in finally 1
in try 4
in finally
>>>
```

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) Rising Exception:-

* Creating exception class object by using rise keyword is known as a rising exception

3) Handling the rised exception:-

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

Guessing a number program

```
class ValueTooSmallError(Exception):
    pass
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 (ValueTooLargeError):
    print ("This value is too large, try again!")
print ("Congratulations! you guessed it correctly")
print ("end")
```

Op:-

```
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.
>>>
```

* 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

```
def jpeg_res(filename):
    """ This functions 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")
```

* 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)
```

* Python program to multiply a matrix

```
x = [[12,7,3],
     [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.casefold()
rev_str = reversed(my_str)
if list(my_str) == list(rev_str):
    print("It is palindrome")
else:
    print("It is not palindrome")
```

* python program to add two matrices

```
x = [[12,7,3],
     [4,5,6],
     [7,8,9]]
y = [[5,8,1],
     [6,7,3],
     [4,5,9]]
result = [[0,0,0],
          [0,0,0],
          [0,0,0]]
for i in range(len(x)):
    for j in range(len(x[0])):
        result[i][j] = x[i][j] + y[i][j]
for r in result:
    print(r)
```

* python program to transpose a matrix

```
x = [[12,7],
     [4,5],
     [3,8]]
result = [[0,0,0],
          [0,0,0]]
for i in range(len(x)):
    for j in range(len(x[0])):
        result[j][i] = x[i][j]
for r in result:
    print(r)
```

```
        else:
            print ("{0} is not a leap year".format(year))
    else:
        print ("{0} is a leap year".format(year))
else:
    print("{0} is not a leap year".format(year))
```

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

```
dec = 344
print ("The decimal value of ", dec, " is:") .
print (bin(dec), "in binary.")
print (oct(dec), "in octal")
print (hex(dec), "in heradecimal")
```

* How to convert the ASCII value

```
c = 'p'
print ("The ASCII value of '" + c + "'", "is", ord(c))
```

* How to shuffle deck of card

```
import itertools, random
deck = list(itertools.product(range(1, 14), ['Spade', 'Heart',
                                'Daimond', 'club']))
random.shuffle(deck)
print ("you got:")
for i in range(5):
        print(deck[i][0], "of", deck[i][1])
```

```
y = temp
print ('The value of x after swapping : { }'.format(7))
print ('The value of Y after swapping : { }'.format(y))
```

* To generate a random number

```
import random
print (random.randint(0,9))
```

* To convert kilometers to miles

```
Kilometers = 5.5
Kilometers = input ("Enter value in kilometers")
conv-fac = 0.621371
miles = Kilometers * conv-fac
print ('%0.3f kilometers is equal to %0.3f miles' %(Kilometers,
                                                    miles))
```

* To convert celsius to faranhiet

```
celsius = 37.5
fahrenheit = (celsius * 1.8) + 32
print ('%0.1f degree celsius is equal to %0.1f degree
        fahrenheit' %(celsius,fahrenheit))
```

* To check a leap year

```
year = 2000
if (year % 4) == 0:
    if (year % 100) == 0:
    *   if (year % 400) == 0:
            print ("{0} is a leap year".format(year))
```

```
        hcf = i
    return hcf
print ("The HCF of ", num1, "and ", num2, "is ", compute HCF(num1,
                                                    num2))
```

* To calculate the Area of triangle

```
a = int ("input enter a number")
b = int ("input enter a number")
c = int ("input enter a number")
s = (a+b+c)/2
area = (s*(s-a)*(s-b)*(s-c)**)** 0.5
print ("The area of triangle is %0.2f" %area)
```

* To solve quadratic equation

```
import cmath
a = 1
b = 5
c = 6
d = (b**2) - (4*a*c)
sol1 = (-b - cmath.sqrt(d))/(2*a)
sol2 = (-b + cmath.sqrt(d))/(2*a)
print ("The solution are {0} and {1}".format(sol1, sol2))
```

* To swap two variables

```
x = 5
y = 10
temp = x
x = y
```

```
    else:
        greater=y
    while (true):
        if ((greater %x==0) and (greater %y==0)):
            lcm= greater
            break
        greater +=1
    return lcm
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 compute HCF(x,y):
    if x>y:
        smaller=y
    else:
        smaller=x
    for i in range(1, smaller+1):
        if ((x%i==0).and (y%i==0)):
```

* fibonacci sequence using recursive functions

```
def recur_fibo(n):
    if n<=1:
        return n
    else:
        return (recur_fibo(n-1)+recur_fibo(n-2))
nterms = 16
if nterms <= 0:
    print (" please enter a positive integer")
else:
    print ("fibonacci sequence:")
    for i in range(nterms):
        print (recur_fibo(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
```

Program.

```
nterms = 10
n1 = 0
n2 = 1
count = 0
if nterms <= 0:
    print("please enter a positive integer")
elif nterms == 1:
    print("Fibonacci sequence upto", nterms, ":")
    print(n1)
else:
    print("Fibonacci sequence upto", nterms, ":")
    while count < nterms:
        print(n1, end=' , ')
        nth = n1 + n2
        n1 = n2
        n2 = nth
        count += 1
```

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 preceeding two terms we then interchange the variables & continue with the process

* 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 preceeding two numbers. This means to say nth term is the sum of (n-1)th and (n-2)th terms.

* 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(num, "is an Armstrong number")
else:
    print(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)
```

* 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 %0.3f is %0.3f' %(num, num_sqrt))
```

for complex numbers:

```
import cmath
num = 1+2j
num_sqrt = cmath.sqrt(num)
print ('The square root of {0} is {1:0.3f}+{2:0.3f}j'.format
                    (num, num_sqrt.real, num_sqrt.imag))
```

* 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.

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 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 = input("enter a number")
    factorial = 1
    if num < 0:
        print("sorry, factorial does not exict 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)
```

Oops principles:-

Oops principles are nothing but certain rules or guidelines.

* Different oops principle are
  1) Encapsulation
  2) Inheritance
  3) Polymorphism
  4) Abstraction.
* 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
PHYTHON
.
.
.

* If we implement the business applications in python language according to procedure orientation it looks like as follows.

```
cname = None
cadd  = None
cacno = None
cbal  = None
ename = None
eadd  = None
eid   = None
esal  = None
```

→ Global Variables.

```
def deposit ( ):
    - - - - -
    - - - - -
    - - - - -
def withdraw ( ):
    - - - - -
    - - - - -
    - - - - -
def transfer ( ):
    - - - - -
    - - - - -
    - - - - -
def baleng ( ):
    - - - - -
    - - - - -
    - - - - -
def da ( ):
    - - - - -
    - - - - -
    - - - - -
def hra ( ):
    - - - - -
    - - - - -
def pf ( ):
    - - - - -
    - - - - -
    - - - - -
def tax ( ):
    - - - - -
    - - - - -
    - - - - -
def tsal ( ):
    - - - - -
    - - - - -
    - - - - -
```

* In procedure oriented 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 a Encapsulation.

Ex:-

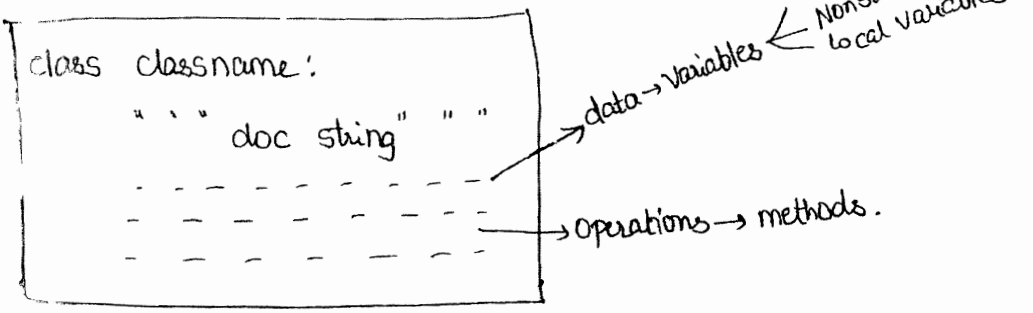
```
cname = None
cadd = None
cacno = None
cbal = None
def deposit ( ):
    - - - - -
    - - - - -
    - - - - -
def withdraw( ):
    - - - - - -
    - - - - - -
    - - - - - - -
def transfer ( ):
    - - - - - -
    - - - - - -
    - - - - - -
def baleng ( ):
    - - - - - -
    - - - - - -
    - - - - - -

ename = None
eadd = None
eid = None
esal = None
def da( ):
    - - - - - -
    - - - - - -
    - - - - - -
def hra ( ):
    - - - - - -
    - - - - - -
    - - - - - -
def pf ( ):
    - - - - - -
    - - - - - -
    - - - - - -
```

```
def tax ( ):
    - - - - -
    - - - - -
    - - - - -
def tsal ( ):
    - - - - -
      - - - -
      - - - -
```

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 it the object is created with original address. that variable is called as pointer variables.
* If you create an object it the object is created with indirect address of the object is called reference variables. Python interpretor 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 interpretor 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 put the data into the object, we can get the data from the object and we 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()
```

```
x2 = x()
print (x2)
x2.m1()
x2.m2()
print ("end")
```

Op:-

```
begin
<__main__.x object at 0x01FECC10>
in m1 of x
in m2 of x
in m1 of x
<__main__.x object at 0x02A9C70>
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 in 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.

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
>>>
```

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
>>>
```

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
>>>
```

Wherever 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)
```

```
x1=X()
x1.m1()
x1.m2()
x2=X()
x2.m2()
```

o/p:- 1000
2000
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 blw methods & constructors

Methods	Constructor
1) method name can be any name	1) Constructor name should be --init--
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 times	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")
```

```
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
>>>
```

Ex:-
```
class X:
    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.

* For local variables of a method memory will be allocated 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()
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
    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(cust.cbname)
c1 = cust("navana", "colombo", 1001, 100000.00)
print(c1)
c1.deposit(20000.00)
c1.withdraw(40000.00)
c1.display()
c2 = cust("sita", "hyd", 1002, 1000.00)
print(c2)
c2.deposit(3000.00)
c2.withdraw(2000.00)
c2.deposit(4000.00)
c2.display()
```

O/P:-

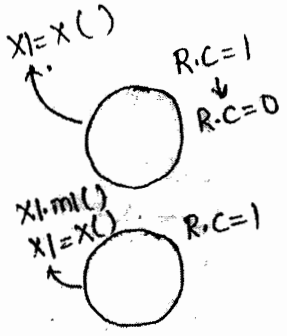
```
<__main__.cust object at 0X023A9D50>
navana
colombo
1001
80000.0
sbi
<__main__.cust object at 0X01FFCD50>
sita
hyd
1002
6000.0
sbi
>>>
```

Garbage Collection:-

* The concept of removing unused or unreference objects from the memory location is known as a Garbage collections.
* 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 interpretor 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 interpretor calls the garbage collector & it removes that object from memory location.

Destructor:-

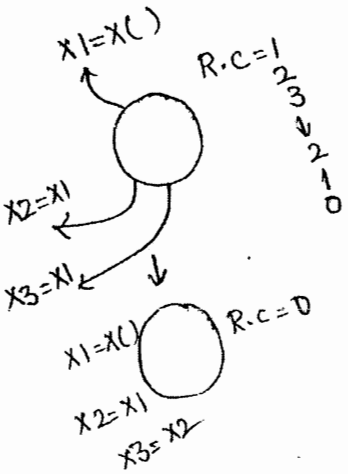
Destructor is one special method and the destructor name should be "__del__".

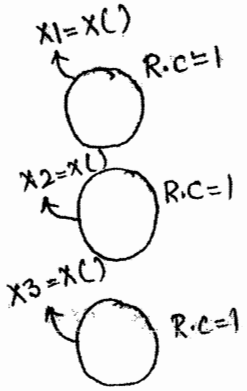
* Destructor will be executed automatically, whenever object is deleting from memory location.

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()
```

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 0X01FDC0D0>
in m1 of X
>>>
```

Ex:-





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)
```

o/p:-
```
in constructor of X
<__main__.X object at 0X023A8DDD>
<__main__.X object at 0X023A8DDD>
<__main__.X object at 0X023A8DDD>
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

i:-

* 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
```

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
>>>
```

Ex:-

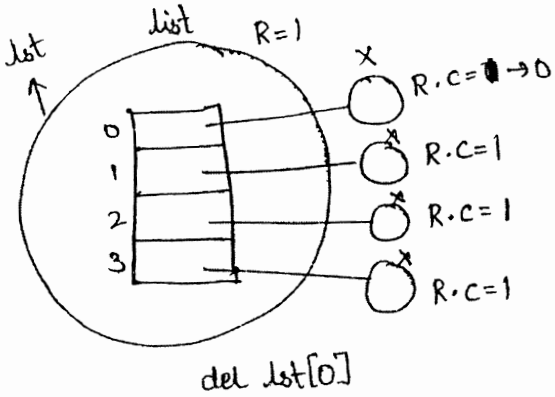
```
Class Emp:
    """ emp app """
    empcount = 0
    def __init__(self, ename, eadd, eid, esal):
        self.ename = ename
        self.eadd = eadd
        self.eid = eid
        self.esal = esal
        Emp.empcount = Emp.empcount + 1
    def display(self):
        print(self.ename, self.eadd, self.eid, self.esal)
    def __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)
```

Op:-

```
scott dallas 7788 3000.00
blake mumbai 7902 4000.00
Smith hyd 7369 3500.0
Total employees are: 3
Total employees are: 2
>>>
```

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
in constructor of X
in constructor of X
in constructor of X
in destructor of X
in destructor of X
in destructor of X
in destructor of X
```



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)
```

O/P:- 1000
1000
2000
>>>

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)
        print(x.b)
x1 = X()
x1.display()
del x.b
x2 = X()
x2.display()
```

O/P:- 1000
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.

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()
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
```
Attribute error: 'X' object 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.

del 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
>>>
```

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

② Is a relationship

① Has a relationship:- The concept of using the properties of one class into another class by using classname 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
>>>
```

IS a relationship (or) Inheritance :-

* The concept of using the properties of one class into another class directly is known as a 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()
```

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 subclass reference variable.

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
>>>

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()
```

o/p:-
```
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__)
```

```
y1 = y()
y1.m1()
y1.m2()
a = y1.__hash__()
print(a)
```

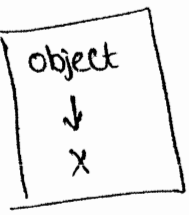
o/p:- (<class '__main__.x'>,)
(<class 'object'>,)
in m1 of x
in m2 of y
2087443
>>>

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 0x023AC190>
<__main__.x object at 0x023AC190>
>>>

Types of inheritances:-

Single Inheritance:- The concept of inheriting the properties from only one class into another class is known as a single inheritance

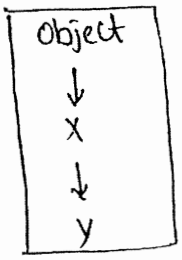


Ex:-
```
class x(object):
    def m1(self):
        print("in m1 of x")
x1=x()
x1.m1()
a=x1.__hash__()
print(a)
```

o/p:-
```
in m1 of x
2337815
>>>
```

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

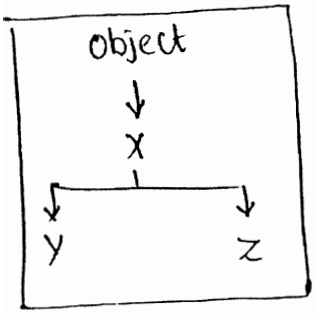


Ex:-
```
class x(object):
    def m1(self):
        print("in m1 of x")
class y(x):
    def m2(self):
        print("in m2 of y")
y1=y()
y1.m1()
y1.m2()
a=y1.__hash__()
print(a)
```

o/p:-
```
in m1 of x
in m2 of y
2337815
>>>
```

Hierarchial Inheritance :-

* The concept of inheriting the properties from one class into multiple classes separately is known as hierarchial inheritance



Ex:-

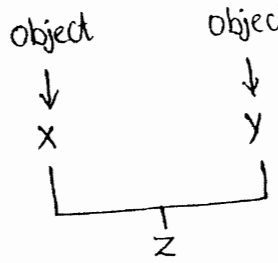
```
class x(object):
    def m1(self):
        print("in m1 of x")
class y(x):
    def m2(self):
        print("in m2 of y")
class z(x):
    def m3(self):
        print("in m3 of z")
y1=y()
y1.m1()
y1.m2()
a=y1.__hash__()
print(a)
z1=z()
z1.m1()
z1.m3()
b=z1.__hash__()
print(b)
```

o/p:-

```
in m1 of x
in m2 of y
2337815
in m1 of x
in m3 of z
2099735
>>>
```

Multiple Inheritance:-

* The concept of Inheriting the properties from multiple classes into single class at a time is known as multiple inheritance.



Ex:-
```
class X(object):
    def m1(self):
        print("in m1 of X")
class Y(object):
    def m2(self):
        print("in m2 of Y")
class Z(X,Y):
    def m3(self):
        print("in m3 of Z")
z1 = Z()
z1.m1()
z1.m2()
z1.m3()
b = z1.__hash__()
print(b)
```

Olp:-
```
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):
```

```
        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 properties from subclass to superclass is known as a '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

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 overwriting 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)
```

o/p:- in two param m1 of X

>>>

* Wherever we define multiple methods with same name within a class then by default python interpretor recognizes 'last defined method' only

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(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
```

```
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.__init__(123)
        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.

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 overwrite 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()
```

o/p:- 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()
```

```
        print("in no param m2 of y")
y1 = y()
y1.m2()
```

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
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
>>>
```

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
>>>
```

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)
```

```
y=sorted(x, key=lambda c:c.cbal, reverse=True)
for q in y:
    print(q)
```

O/P:-

```
ravana  colombo  1001  100000.00  sbi
sita    hyd      1002  1000.0     sbi
miller  dallas   1003  3000.0     sbi
ravana  colombo  1001  100000.00  sbi
sita    hyd      1002  1000.00    sbi
miller  dalas    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 a 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 m1(self):
        print(X.__a)

x1=X()
x1.m1()
print(X.__a)
```

O/P:-

```
1000
Attribute Error
>>>
```

Implementing Inheritance:-

* Hidden properties of a class cannot be accessed directly within its subclass also.

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(Y.__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:
    __a = 1000
    def m1(self):
        print("in m1 of X")
X1 = X()
print("in m1 of X")
```

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")
```

```
X1 = X()
print(X1)
X1.m1()
X1.m2(1000)
X2 = X()
print(X2)
X2.m1()
X2.m2(2000)
```

Op:-

```
<__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)
```

```
        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)
```

O/P:-

```
<__main__.x object at 0x012BAC30>
<__main__.x object at 0x012BAC30>
in m1 of x
<__main__.x object at 0x012BAC30>
1234
in m2 of x
<__main__.x object at 0x0221E0D0>
<__main__.x object at 0x0221E0D0>
in m1 of x
<__main__.x object at 0x0221E0D0>
5678
in m2 of x
>>>
```

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.
* Online transaction processing (OLTP) operations implementations purpose we have to use ORDBMS.
* Different ORDBMS are

  ORACLE
  MYSQL
  DB2
  SQLITE 3
  SQLSERVER
  TERA DATA
  :

* To provide the communication blw 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, 11G, 12C } Enterprise Edition

* Download & install the oracle database 11g or 12C Enterprise edition.

* Username = SCOTT
  password = tiger
  Servicename (or) logical database name (or) SID or host storing
  name = orcl
  port no = 1521
  IP address = localhost
* 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',
'ipadd of the computer where database is installed : portno/
databaseservice 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 sqlqueries 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()
```

* By calling method of cursor objects, we can 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()
```

o/p:- (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 occured")
Finally:
    con != None:
    con.close()
    print("connection is closed")
```

O/P:- connection is established
error occured
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()
```

o/p:- (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 eventhough we execute "rollback" command, the modified records will not come back to its original state
* Create the empl table in the database by using following command

Create table empl 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 empl 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 = str(input("enter sacno"))
    tamt = str(input("enter transfer amt"))
    query1 = "update cust set cbal = cbal - "+tamt+" where cacno="
    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+" +tamt+ " where cacno ="

        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()
```

To create table & insert data into table

```
SQL> create table cust (cacno(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 sacno 1001
enter transfer amt 2000.00
enter dacno 1002
>>>>
```

(or) enter sacno 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
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)
>>>
```

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.setinputsizes(int, 10)
cur.executemany("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()
```

Op:-

```
(1, 'First')
(2, 'second')
(3, 'Third')
(4, 'Fourth')
(5, 'Fifth')
(6, 'sixth')
(7, 'seventh')
>>>
```

```
Create table mytab
Create table mytab (id number(1),
        varchar2(10));
```

Creation of a table for bigtab:-

```
SQL> create table bigtab (mycol varchar2(20));
```

Table created.

Cursor array size:-

* 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()
```

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. /.

Ex1:- Procedure creation program:-

```
Create or replace procedure P1(a in number, b out number)
is
begin
b: a*a;
end P1;
/
```

* 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
 varchar2, cnt 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

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;
/
```

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()
```

```
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 ()
```

o/p:- ('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 = "lokesh.mysql.pythonanywhere-services.com",
user = "lokesh", passwd = "sathya1234", db = "lokesh$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")
```

o/p:- connection established
<MYSQLdb.cursors.cursor object at 0X7ffa00ed2be0>
(1, 'django')
(2, 'flask')
connection closed
>>>

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 ()
```

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()
```

* 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, 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()
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.

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()
```

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()
```

Infinite loop with time

Ex:-
```
import time
x=1
While True:
    print(x)
    x=x+1
    time.sleep(1)
```

* 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)
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.

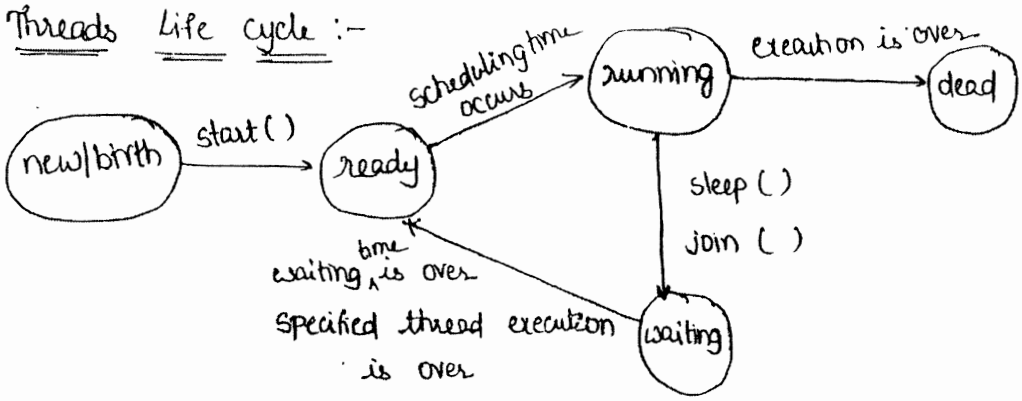
Ex:-

```
import threading
import time
class X (threading.Thread):
    def run(self):
        lc.acquire()
        myfunc("django")
        lc.release()
class Y (threading.Thread):
    def run(self):
        lc.acquire()
        myfunc("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()
```

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 fun (or) 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__ then 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
    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)
print(next(It1))
print(next(It1))
print(next(It1))
print(next(It1))
print(next(It1))
print(next(It1))
```

O/P:-
```
1
2
4
8
16
32
raise stop Iteration
```

Ex1:-
```
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)
```

```
while True:
    try:
        print (next (it1)):
    except (Stop iterations):
        break
```

O/P:
```
1
2
4
8
16
32
>>>
```

* 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 form 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.

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)
```

o/p:

```
1
2
4
8
16
32
>>>
```

Ex2:

```
x = [10, 20, 30, 40]
it = iter(x)
```

```
print (next (it))
print (next (it))
print (next (it))
print (next (it))
print (next (it))
```

o/p:

```
10
20
30
40
stop iteration exception is raised
>>>
```

Ex:

```
x = [10, 20, 30, 40]
it = iter(x)
while True:
    try:
        print (next (it))
    except (StopIteration):
        break
```

o/p:

```
10
20
30
40
>>>
```

Ex:

```
x = [10, 20, 30, 40]
for p in x:
    print(p)
```

O/P:

```
10
20
30
40
>>>
```

Ex:

```
class x:
    def m1(self):
        print("in m1 of x")
    for p in x():
        print(p)
```

O/P:

Type error

Ex:

```
class Infilter:
    """ Infinite iterator to return all old numbers """
    def __iter__(self):
        self.num = 1
        return self
    def __next__(self):
        num = self.num
        Self.num += 2
        return num
```

```
i = Infilter()
it = iter(i)
while True:
    try:
        print(next(it))
    except(StopIteration):
        break
```

Dn 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 ie., 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 it 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 execution.
* 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

Ex.

```
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))
```

Olp.

```
<generator object my_gen at 0X024B7180>
This is printed first
1
This is printed second
2
This is printed last
3
'Stop iteration exception raised'
```

Ex2,

```
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
y
.
.
.
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))
```

```
        except(StopIteration):
            break
```

E1:

```
n7.py
h
s
e
k
o
l
>>>
```

E1:

```
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)
```

o/p:

```
n7.py
h
s
e
k
o
l
>>>
```

E1: By using tuple comprehension

```
x = (p for p in range(5))
print(x)
for i in x:
    print(i)
```

Infinite Generator

Ex:

```
def all_even():
    n=0
    while True:
        yield n
        n+=2
for p in all_even():
    print(p)
```

Ex:

```
def PowTwoGen(max=0):
    n=0
    while n<=max:
        yield 2**n
        n+=1
for a in PowTwoGen(5):
    print(a)
```

Olp:

```
n10.py
1
2
4
8
16
32
>>>
```

* Generator implementation is simple when compared to the iterator implementation.
* Generators are memory efficient.
* By using Generators we can represent the infinite strings.

Closures

Nested Functions:

* A function which is defined inside 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
>>>
```

* We can call the inner function of enclosing function, outside of that enclosing function by returning inner function object through the outer function

Ex

```
def outer():
    print("in outer")
    def inner():
        print("in inner")
    return inner
myfunc = outer()
myfunc()
```

O/P:

```
in outer
in inner
>>>
```

* 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

Ex:

```
def outer():
    print("in outer")
    def inner(a):
        print(a)
        print("in inner")
    return inner
myfunc = outer()
del outer
myfunc(1000)
outer()
```

o/p:

```
in outer
1000
in inner
error
>>>
```

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.

Ex:

```
def outer(a):
    print("in outer")
    def inner():
        print(a)
        print("in inner")
    return inner
myfunc=outer(1000)
myfunc()
```

o/p:

```
in outer
1000
in inner
>>>
```

Ex:

```
def outer(a):
    print("in outer")
    def inner():
        print(a)
        print("in inner")
    return inner
my_func = outer(1000)
del outer
my_func()
```

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

Ex:

```
def make_multiplier_of(n):
    def multiplier(x):
        return x*n
    return multiplier
times3 = make_multiplier_of(3)
print(times3(5))
```

O/P:

```
15
>>>
```

Decorators.

→ Decorators are used to add the 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 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")
        fun()
        return inner
@ make_pretty
    def ordinary():
        print("I am ordinary")
    ordinary()
```

o/p:

```
I got decorated
I am ordinary
```

Ex2.

```
def smart-pretty(func)
    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,5)
divide(10,5)
```

o/p:

I am going to divide 10, and 5

2.0

19th Aug-17

Regular-Expressions:-

Regular-expressions are nothing but special characters and by using regular-expressions special characters we can define the regular expression Pattrens.

→ By using Regular-expression pattrens we can extract the required information from the given data. We can Perform the data-format validation and we can define the url pattrens for the webpages.

Regular-expressions characters

1. * → it matches 0 or more accurances of its other preceding characters.

Ex:-

ab*c
abc
abbc
abbbc
abbbbbc

+ →

2. it matches one/more occurences of its preceding character.

Eg 1: ab+c

Ac # error

Abc

Abbc

Abbbbbc

3. ? → it matches zero/one occurence of its preceding character

Eg 1: ab?c

Ac

Abc

Abbc # error

Eg: ②

Pea?rl

Perl

Pearl

③

colou?r

color

colour

4. . → it matches any single character.

Eg 1. a.c

A$C

A C

Abdc # error

Aqc

5. [] → it matches any single character in the given

Eg 1: b[aeiou]d

Bad

Bed

Bid

Bud

6. [^] → it matches any single character other than in the given list.

eg 1: b[^aeiou]d

Bad # error

Bid # error

Bud # error
Bed # error
B8d

Bpd

7. [start-end] → it matches any single character in the given range.

eg 1: x[a-e]y

xay
xcy
xey
xfy } error
xpy }

2. [0-9] → any single digit
3. [a-z] → any one lowercase alphabet
4. [A-Z] → any one uppercase alphabet
5. [a-zA-Z] → any one alphabet
6. [a-zA-Z0-9] → any one alphanumeric
7. [^0-9] → any single non digit
8. [^a-z] → any one non lowercase alphabet
9. [^A-Z] → any one non uppercase alphabet
10. [^a-zA-Z] → any one non alphabet

8. (|) → match only any one string in the list.

eg1: (java|hadoop|python)

9. {m} → it matches exact occurence of its preceding character.

eg 1: Ab{3}c → b repeats 3 times

Abbc # error

Abbbc

Abbbbc # error

10. {m,n} → it matches min 'm' occurences and max 'n' occurences of its preceding characters.

eg 1: Ab{3,5}c

Abbc # error

Abbbc

Abbbbc

Abbbbbbbbc # error

11. {m,} → it matches min m occurences & max no limit of its preceding character.

eg 1: Ab{3,}c

Abbbc

Abbc # error

Abbbbbbbc

except

12. ^ → start of the line

eg1: ^per| . , ^[abc] , ^[^abc]

13. $ → end of the line

eg 1: perl$, [0-9]$ → single digit

14. \d or [0-9] → any single digit

eg:- [0-9][0-9][0-9][0-9] or [0-9]{4} or \d\d\d\d or \d{4}

15. \D or [^0-9] → any single non digit

16. \w or [a-zA-Z0-9] → any alphanumeric

17. \W or [^a-zA-Z0-9] → any non alphanumeric or specia

18. \s => ' ', \t, \n

19. \b → word boundary

By using Regular-Expressions special character we can define the regular expressions patterns.

→ the sequence of regular-expression characters which represented in " " and it is preceded by the 'r' character is known as a regular-expression Pattren.

→ by calling predefined func/methods of R.E module we can extract the pattren matching information from the given data.

eg:-①

```
import re
regex = r"[a-zA-Z]+\d+"
data = "June 21, Jan 19, Nov 16, 14 July 21 Dec 31"
matches = re.findall(regex, data)
for match in matches:
    Print(match)
```

o/p

June 21
Jan 19
Nov 16
July 21
~~Dec~~

eg:-②

```
import re
regex = r"([a-zA-Z]+)\d+"
matches = re.findall(regex, "June 24, August 19,
                    Dec 12, Jul")
for match in matches:
    Print("Match month:", match)
```

o/p

match month : June
: August
: Dec

Eg:-

```
import re
regex = r"[a-zA-Z]+\d+"
matches = re.finditer(regex, "June 24, August 9 De
12,
for matches in matches:
        Print ("match at index:", match start(),
                                    match end())
```

O/P

Match at index : 0 7
: 9 17
: 8 24

Eg:-

emp.ext file

1001, scott, 2000.00
miller, 5000.00, 1002
3000.00, 1003, blake

```
import re
regex = r"[a-zA-Z]+"
x= open("emp.txt")
data = x.read()
matches = re.findall (regex, data)
for match in matches:
        Print (match)
```

11th sept

```
import re
regex = re.compile(r"(\w+) World")
result = regex.search("python world hello world python is easiest")
if result:
    print(result.start(), result.end())
```

o/p :-

13, 24

Note: searching for only one.

* Create Mobile number

```
import re
regex = re.compile(r"\d{10}")
while True:
    mobno = input("enter mobno")
    if len(mobno) == 10:
        result = regex.search(mobno)
        if result:
            print("entered mob no is valid")
            break
        else:
            print("mobile no should contain 10")
    else:
        print("in valid mobile no it should
```

10 digits")

olp:-

Enter mobileno 123456=8901
invalid mobile no it should contain 10 digits
enter mobno abcde123
mobile no should contain digits only
enter mobno 1234567890
enter mob no is valid.

* Pwd formate Validation

```
import re
regex = re.compile(r"(\w+) World")
result = regex.findall("Hello World, Bonjour World")
for result in result1:
    Print(result)
```

olp:-

```
Hello
Bonjour
>>>
```

```
import re
regex = re.compile(r"(\w+) World")
Print(regex.sub(r"\1 Earth", "Hello world, new world")
```

olp

```
Hello Earth,
        new Earth
```

Eg:-

```
import re
regex = r"\d{3}.\d{2}.\d.(\d{2}$)"
da = re.findall(regex, "223.12.1.10")
for ip in da:
    Print(ip)
```

o/p 10

* Command line Arguments:

The concept of passing the value from the command prompt at the time of running the Python file is known as a "command line Arguments".

→ Python file name is also taken as a one of the command line argument.

→ All the command line arguments are stored into the list object in the form of strings; and that list object is stored into the "argv" variable of "sys" module.

→ SYS is a one predefined module.

Ex:-

```
import sys
Print (sys.argv)
Print (len (sys.argv))
for p in sys.argv:
    Print (p)
```

o/p →

- Save the above file with cargs1.py; in c:\python 36-32 folder.
- Now open the command & execute following commands.

```
c:\python 36-32 > python cargs1.py ↵
o/p
    ['cargs1.py']
    1
    cargs1.py
c:\python 36-32 > python cargs1.py sathya lokesh
                                    python
```

o/p

['cargs1.py', 'sathya', 'lokesh', 'python']

4

cargs1.py

sathya

lokesh

python

c:\python 36-32 > python cargs1.py 100 200 300 400

o/p ['cargs1.py', '100', '200', '300', '400']

5

cargs 1.py

100

200

300

400

ex:-

```
import sys
if len(sys.argv) == 3:
    try:
        x = int(sys.argv[1])
        y = int(sys.argv[2])
        z = x+y
        Print("addition of", x, "and", y, "is:",
```

```
except (value-error):
    Print (" enter numerical arguments only")
else:
    Print (" enter two userdefined argument only")
```

o/p

Save this file as cargs.py

open cmd and run following commands

c:\python 36-32 > python cargs.py 100 abc

o/p → enter numerical arguments only.

c:\python 36-32 > python cargs.py 100 200 300 ↵

o/p → enter two userdefined arguments only

c:\python 36-32 > python cargs.py 123 456

o/p → addition of 123 and 456 is : 579

Main Module (__main__):-

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

→ The name of the current module is stored in one Predefined attribute called __name__.

Ex: demo.py

```
if __name__ == "__main__":
    i = input ("enter fno")
    j = input ("enter sno")
    try:
        x = int(i)
        y = int(j)
        z = x+y
        Print ("addition of", x "and", y, "is:" z)
    except (value Error):
        Print ("enter numerical argument only")
```

directly run →

```
enter fno 123
enter sno 459
addition of 123 & 456 is: 579
```

test.py

```
import demo
Print ("hi from test")
```

o/p hi from tat

O.S. module

O.S module is a predefined module and which Provides various functions and methods to perform the operating system related activities, such as creating the files, removing the files, creating the directories removing the directories, executing the operating sly related commands, etc.

Ex:-

```
import os
cwd = os.getcwd()
Print ("1", cwd)
os.chdir ("samples")
Print ("2", os.getcwd())
os.chdir (os.pardir)
Print ("3", os.getcwd())
```

O/P →

```
D:\python\studmat\osmodule
D:\python\studmat\osmodule\sample
D:\python\studmat\osmodule
```

Ex:-

```
import os
Var= os.listdir ("samples")
```

```
for file in var:
    Print (file)
```

O/P →

```
mydir
sample.tmp
sample.txt
```

Ex:-

```
import os
os.makedirs("test/multiple/levels")
fp= open("test/multiple/levels/myfiles", "w")
fp.write("python\nlokesh\nsathya")
fp.close()
# os.remove ("test/multiple/levels/myfile")
# os.removedirs ("test/multiple/levels")
```

O/P → directories will be created.

Ex- create directory samples/sample.txt

Ex:-

```
import os
import string
def replace (inpfile, search-for, replace,
```

```
fi = open(inpfile)
fo = open("samples/newfile.txt", "w")
for s in fi.readlines():
    fo.write(string.replace(s, search for
                            replace_with))
fi.close()
fo.close()
string.replace("samples/sample.txt", ~~txt~~ "sathya", "lokesh")
```

o/p →

check the directories and folder

```
help(str.replace) ↵
```

```
[training@localhost ~] $
                    $ nanoDemo.py
```

* Write the following code in the oscmd.py file:-

```
import os
if os.name == "int":
    command = "dir"
else:
    command = "pwd"
```

os.system(command)

Press → ctrl+X ↵

Prs → y

- nano oscmd.py ↵
- python oscmd.py ↵

o/p

/home/training

on newVersion

ex:-

```
import os
def replace(inpfile, search_for, replace_with):
    fi = open(inpfile)
    fo = open("samples/newfile.txt", "w")
    for s in fi.readlines():
        fo.write(s.replace(search_for, replace_
                           with))
    fi.close()
    fo.close()
replace("samples/samples.txt", "lokesh", "sathya
```

osdemo.py

```
import os
if os.name == "nt":
    command = "dir"
else:
    command = "pwd"
os.system(command)
```

Run on → execute the above program from the cmd.

c:\user\lokesh > python D:\6pm python\Test.py

c:\user\lokesh > python D:\python\osmodule\osdemo.py

- open the vmware workstation and start the centos.
- open the terminal

[training @ localhost ~] $ nano Demo.py ↵
rm Demo.py

The following files has the usage information for two buildings:

- building #1 and building #2.
- For 5 weeks – week 1 to week 5
- For 5 days in a week
- From 9AM to 17hrs (5PM) every day

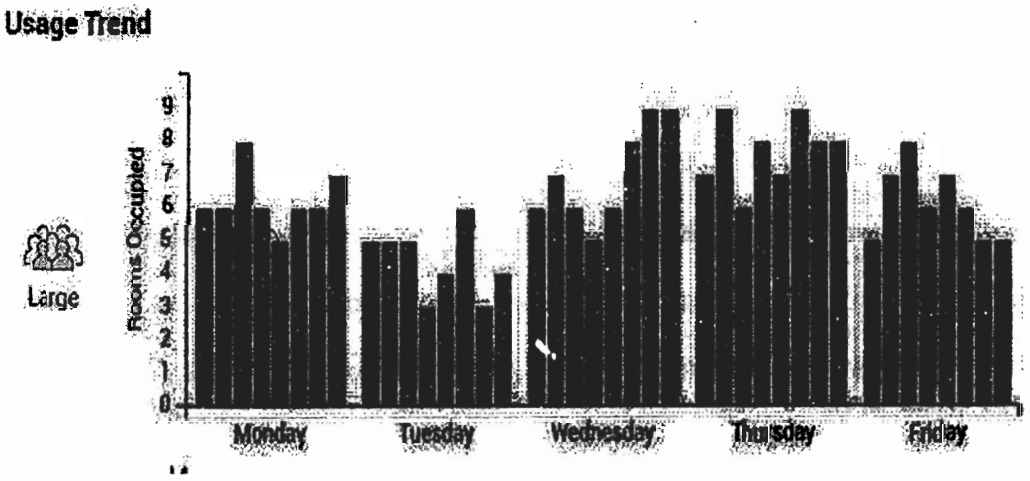
This is a JSON output from an Enlighted application.


```
[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-
count":{"Small":9,"Phone Booth":0,"Medium":5,"Large":4}}],
```


- For the line above it shows that on Wednesday at 2PM (14:00hrs),
  - 9 small conference rooms were booked
  - 0 Phone booths were used
  - 5 Medium conference rooms were used
  - 4 large conference rooms were used,

Please provide a script (or scripts) that do(es) the following:

a. Plot how many large conference rooms were booked over the day during each 5 day for all 5 weeks. Showing the example graph for 1 week – each column represents how many rooms were used at that hour during that day. You will have 1 such graph for every week or a combined graph.



b. Provide a graph that compares the usage between the two buildings for any room category – by looking at your graph we can see which building is more occupied for say large conference rooms.

B1_W1.txt

[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-count":{"Small":5,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Thursday"},{"time":"11:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":2,"Large":0}}],[{"dow":"Friday"},{"time":"09:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":2,"Large":1}}],[{"dow":"Friday"},{"time":"11:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":1,"Large":3}}],[{"dow":"Tuesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":1,"Large":1}}],[{"dow":"Friday"},{"time":"15:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":0,"Large":2}}],[{"dow":"Monday"},{"time":"12:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":5,"Large":3}}],[{"dow":"Monday"},{"time":"14:00"},{"conference-categories-
count":{"Small":8,"Phone
Booth":0,"Medium":5,"Large":3}}],[{"dow":"Thursday"},{"time":"09:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":1,"Large":1}}],[{"dow":"Monday"},{"time":"10:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Monday"},{"time":"16:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":6,"Large":3}}],[{"dow":"Monday"},{"time":"15:00"},{"conference-categories-
count":{"Small":8,"Phone
Booth":0,"Medium":7,"Large":3}}],[{"dow":"Tuesday"},{"time":"13:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":1,"Large":3}}],[{"dow":"Tuesday"},{"time":"15:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":3,"Large":2}}],[{"dow":"Wednesday"},{"time":"11:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Thursday"},{"time":"14:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":2,"Large":1}}],[{"dow":"Thursday"},{"time":"16:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":2,"Large":3}}],[{"dow":"Tuesday"},{"time":"11:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":2,"Large":2}}],[{"dow":"Friday"},{"time":"12:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":2,"Large":3}}],[{"dow":"Monday"},{"time":"09:00"},{"conference-categories-
count":{"Small":9,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Wednesday"},{"time":"13:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":5,"Large":3}}],[{"dow":"Wednesday"},{"time":"15:00"},{"conference-categories-

count":{"Small":5,"Phone
Booth":0,"Medium":2,"Large":0}}],[{"dow":"Thursday"},{"time":"10:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":3,"Large":0}}],[{"dow":"Thursday"},{"time":"12:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":2,"Large":2}}],[{"dow":"Friday"},{"time":"10:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":1,"Large":4}}],[{"dow":"Friday"},{"time":"16:00"},{"conference-categories-
count":{"Small":1,"Phone
Booth":0,"Medium":1,"Large":1}}],[{"dow":"Monday"},{"time":"13:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Friday"},{"time":"14:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":3,"Large":2}}],[{"dow":"Monday"},{"time":"11:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Wednesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":2,"Large":2}}],[{"dow":"Tuesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Tuesday"},{"time":"14:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":5,"Large":2}}],[{"dow":"Tuesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Wednesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Thursday"},{"time":"15:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":1,"Large":4}}],[{"dow":"Wednesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":4,"Large":3}}],[{"dow":"Tuesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":3,"Large":1}}],[{"dow":"Wednesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":1,"Large":1}}],[{"dow":"Friday"},{"time":"13:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":1,"Large":3}}],[{"dow":"Thursday"},{"time":"13:00"},{"conference-categories-
count":{"Small":4,"Phone Booth":0,"Medium":4,"Large":1}}]]

B1_ W2.txt

[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":5,"Large":4}}],[{"dow":"Thursday"},{"time":"11:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":6,"Large":1}}],[{"dow":"Friday"},{"time":"09:00"},{"conference-categories-count":{"Small":4,"Phone Booth":0,"Medium":1,"Large":2}}],[{"dow":"Friday"},{"time":"11:00"},{"conference-categories-count":{"Small":6,"Phone Booth":0,"Medium":3,"Large":2}}],[{"dow":"Tuesday"},{"time":"09:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Friday"},{"time":"15:00"},{"conference-categories-count":{"Small":6,"Phone Booth":0,"Medium":1,"Large":1}}],[{"dow":"Monday"},{"time":"12:00"},{"conference-categories-count":{"Small":10,"Phone Booth":0,"Medium":5,"Large":3}}],[{"dow":"Monday"},{"time":"14:00"},{"conference-categories-count":{"Small":10,"Phone Booth":0,"Medium":7,"Large":3}}],[{"dow":"Thursday"},{"time":"09:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":2,"Large":2}}],[{"dow":"Monday"},{"time":"10:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":7,"Large":4}}],[{"dow":"Monday"},{"time":"16:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":9,"Large":4}}],[{"dow":"Monday"},{"time":"15:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":9,"Large":3}}],[{"dow":"Tuesday"},{"time":"13:00"},{"conference-categories-count":{"Small":10,"Phone Booth":0,"Medium":5,"Large":3}}],[{"dow":"Tuesday"},{"time":"15:00"},{"conference-categories-count":{"Small":8,"Phone Booth":0,"Medium":4,"Large":3}}],[{"dow":"Wednesday"},{"time":"11:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":1,"Large":3}}],[{"dow":"Thursday"},{"time":"14:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":4,"Large":3}}],[{"dow":"Thursday"},{"time":"16:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":2,"Large":2}}],[{"dow":"Tuesday"},{"time":"11:00"},{"conference-categories-count":{"Small":8,"Phone Booth":0,"Medium":3,"Large":4}}],[{"dow":"Friday"},{"time":"12:00"},{"conference-categories-count":{"Small":5,"Phone Booth":0,"Medium":3,"Large":1}}],[{"dow":"Monday"},{"time":"09:00"},{"conference-categories-count":{"Small":10,"Phone Booth":0,"Medium":5,"Large":4}}],[{"dow":"Wednesday"},{"time":"13:00"},{"conference-categories-count":{"Small":6,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Wednesday"},{"time":"15:00"},{"conference-categories-

count":{"Small":7,"Phone
Booth":0,"Medium":5,"Large":3}}],[{"dow":"Thursday"},{"time":"10:00"},{"conference-categories-
count":{"Small":10,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Thursday"},{"time":"12:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":2,"Large":1}}],[{"dow":"Friday"},{"time":"10:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":2,"Large":1}}],[{"dow":"Friday"},{"time":"16:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":3,"Large":0}}],[{"dow":"Monday"},{"time":"13:00"},{"conference-categories-
count":{"Small":11,"Phone
Booth":0,"Medium":7,"Large":4}}],[{"dow":"Friday"},{"time":"14:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":3,"Large":0}}],[{"dow":"Monday"},{"time":"11:00"},{"conference-categories-
count":{"Small":12,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Wednesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":1,"Large":3}}],[{"dow":"Tuesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":2,"Large":3}}],[{"dow":"Tuesday"},{"time":"14:00"},{"conference-categories-
count":{"Small":10,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Tuesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":10,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Wednesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":8,"Phone
Booth":0,"Medium":1,"Large":4}}],[{"dow":"Thursday"},{"time":"15:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Wednesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Tuesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":10,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Wednesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Friday"},{"time":"13:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":2,"Large":0}}],[{"dow":"Thursday"},{"time":"13:00"},{"conference-categories-
count":{"Small":6,"Phone Booth":0,"Medium":3,"Large":4}}]]

B1_ W3.txt

[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":7,"Large":4}}],[{"dow":"Thursday"},{"time":"11:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Friday"},{"time":"09:00"},{"conference-categories-count":{"Small":6,"Phone Booth":0,"Medium":3,"Large":2}}],[{"dow":"Friday"},{"time":"11:00"},{"conference-categories-count":{"Small":5,"Phone Booth":0,"Medium":1,"Large":1}}],[{"dow":"Tuesday"},{"time":"09:00"},{"conference-categories-count":{"Small":4,"Phone Booth":0,"Medium":2,"Large":3}}],[{"dow":"Friday"},{"time":"15:00"},{"conference-categories-count":{"Small":6,"Phone Booth":0,"Medium":6,"Large":2}}],[{"dow":"Monday"},{"time":"12:00"},{"conference-categories-count":{"Small":11,"Phone Booth":0,"Medium":4,"Large":4}}],[{"dow":"Monday"},{"time":"14:00"},{"conference-categories-count":{"Small":11,"Phone Booth":0,"Medium":9,"Large":4}}],[{"dow":"Thursday"},{"time":"09:00"},{"conference-categories-count":{"Small":2,"Phone Booth":0,"Medium":5,"Large":4}}],[{"dow":"Monday"},{"time":"10:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":8,"Large":3}}],[{"dow":"Monday"},{"time":"16:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":7,"Large":4}}],[{"dow":"Monday"},{"time":"15:00"},{"conference-categories-count":{"Small":10,"Phone Booth":0,"Medium":7,"Large":4}}],[{"dow":"Tuesday"},{"time":"13:00"},{"conference-categories-count":{"Small":6,"Phone Booth":0,"Medium":3,"Large":4}}],[{"dow":"Tuesday"},{"time":"15:00"},{"conference-categories-count":{"Small":1,"Phone Booth":0,"Medium":6,"Large":4}}],[{"dow":"Wednesday"},{"time":"11:00"},{"conference-categories-count":{"Small":4,"Phone Booth":0,"Medium":4,"Large":4}}],[{"dow":"Thursday"},{"time":"14:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":6,"Large":4}}],[{"dow":"Thursday"},{"time":"16:00"},{"conference-categories-count":{"Small":11,"Phone Booth":0,"Medium":7,"Large":3}}],[{"dow":"Tuesday"},{"time":"11:00"},{"conference-categories-count":{"Small":5,"Phone Booth":0,"Medium":2,"Large":3}}],[{"dow":"Friday"},{"time":"12:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":5,"Large":1}}],[{"dow":"Monday"},{"time":"09:00"},{"conference-categories-count":{"Small":10,"Phone Booth":0,"Medium":7,"Large":3}}],[{"dow":"Wednesday"},{"time":"13:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":6,"Large":4}}],[{"dow":"Wednesday"},{"time":"15:00"},{"conference-categories-

count":{"Small":5,"Phone
Booth":0,"Medium":5,"Large":3}}],[{"dow":"Thursday"},{"time":"10:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Thursday"},{"time":"12:00"},{"conference-categories-
count":{"Small":9,"Phone
Booth":0,"Medium":2,"Large":3}}],[{"dow":"Friday"},{"time":"10:00"},{"conference-categories-
count":{"Small":9,"Phone
Booth":0,"Medium":3,"Large":1}}],[{"dow":"Friday"},{"time":"16:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":1,"Large":1}}],[{"dow":"Monday"},{"time":"13:00"},{"conference-categories-
count":{"Small":12,"Phone
Booth":0,"Medium":9,"Large":3}}],[{"dow":"Friday"},{"time":"14:00"},{"conference-categories-
count":{"Small":8,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Monday"},{"time":"11:00"},{"conference-categories-
count":{"Small":12,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Wednesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Tuesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":5,"Large":3}}],[{"dow":"Tuesday"},{"time":"14:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Tuesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Wednesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Thursday"},{"time":"15:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Wednesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Tuesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Wednesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Friday"},{"time":"13:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":2,"Large":2}}],[{"dow":"Thursday"},{"time":"13:00"},{"conference-categories-
count":{"Small":3,"Phone Booth":0,"Medium":4,"Large":4}}]]

B1_ W4.txt

[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-count":{"Small":5,"Phone Booth":0,"Medium":6,"Large":3}}],[{"dow":"Thursday"},{"time":"11:00"},{"conference-categories-count":{"Small":4,"Phone Booth":0,"Medium":10,"Large":3}}],[{"dow":"Friday"},{"time":"09:00"},{"conference-categories-count":{"Small":5,"Phone Booth":0,"Medium":4,"Large":3}}],[{"dow":"Friday"},{"time":"11:00"},{"conference-categories-count":{"Small":8,"Phone Booth":0,"Medium":5,"Large":3}}],[{"dow":"Tuesday"},{"time":"09:00"},{"conference-categories-count":{"Small":1,"Phone Booth":0,"Medium":2,"Large":1}}],[{"dow":"Friday"},{"time":"15:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":2,"Large":2}}],[{"dow":"Monday"},{"time":"12:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":5,"Large":3}}],[{"dow":"Monday"},{"time":"14:00"},{"conference-categories-count":{"Small":8,"Phone Booth":0,"Medium":6,"Large":3}}],[{"dow":"Thursday"},{"time":"09:00"},{"conference-categories-count":{"Small":4,"Phone Booth":0,"Medium":7,"Large":3}}],[{"dow":"Monday"},{"time":"10:00"},{"conference-categories-count":{"Small":6,"Phone Booth":0,"Medium":7,"Large":2}}],[{"dow":"Monday"},{"time":"16:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":4,"Large":3}}],[{"dow":"Monday"},{"time":"15:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":6,"Large":4}}],[{"dow":"Tuesday"},{"time":"13:00"},{"conference-categories-count":{"Small":10,"Phone Booth":0,"Medium":3,"Large":2}}],[{"dow":"Tuesday"},{"time":"15:00"},{"conference-categories-count":{"Small":10,"Phone Booth":0,"Medium":4,"Large":3}}],[{"dow":"Wednesday"},{"time":"11:00"},{"conference-categories-count":{"Small":10,"Phone Booth":0,"Medium":6,"Large":3}}],[{"dow":"Thursday"},{"time":"14:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":8,"Large":4}}],[{"dow":"Thursday"},{"time":"16:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":6,"Large":2}}],[{"dow":"Tuesday"},{"time":"11:00"},{"conference-categories-count":{"Small":9,"Phone Booth":0,"Medium":6,"Large":4}}],[{"dow":"Friday"},{"time":"12:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":4,"Large":2}}],[{"dow":"Monday"},{"time":"09:00"},{"conference-categories-count":{"Small":5,"Phone Booth":0,"Medium":5,"Large":1}}],[{"dow":"Wednesday"},{"time":"13:00"},{"conference-categories-count":{"Small":5,"Phone Booth":0,"Medium":5,"Large":4}}],[{"dow":"Wednesday"},{"time":"15:00"},{"conference-categories-

count":{"Small":4,"Phone
Booth":0,"Medium":6,"Large":3}}],[{"dow":"Thursday"},{"time":"10:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":7,"Large":3}}],[{"dow":"Thursday"},{"time":"12:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":6,"Large":3}}],[{"dow":"Friday"},{"time":"10:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":4,"Large":1}}],[{"dow":"Friday"},{"time":"16:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":2,"Large":1}}],[{"dow":"Monday"},{"time":"13:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":7,"Large":2}}],[{"dow":"Friday"},{"time":"14:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Monday"},{"time":"11:00"},{"conference-categories-
count":{"Small":8,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Wednesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Tuesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Tuesday"},{"time":"14:00"},{"conference-categories-
count":{"Small":9,"Phone
Booth":0,"Medium":4,"Large":3}}],[{"dow":"Tuesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":10,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Wednesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":9,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Thursday"},{"time":"15:00"},{"conference-categories-
count":{"Small":7,"Phone
Booth":0,"Medium":8,"Large":3}}],[{"dow":"Wednesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":10,"Phone
Booth":0,"Medium":7,"Large":3}}],[{"dow":"Tuesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":9,"Phone
Booth":0,"Medium":3,"Large":1}}],[{"dow":"Wednesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":5,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Friday"},{"time":"13:00"},{"conference-categories-
count":{"Small":8,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Thursday"},{"time":"13:00"},{"conference-categories-
count":{"Small":5,"Phone Booth":0,"Medium":6,"Large":4}}]]

B1_ W5.txt

[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":7,"Large":3}}],[{"dow":"Thursday"},{"time":"11:00"},{"conference-categories-count":{"Small":4,"Phone Booth":0,"Medium":2,"Large":3}}],[{"dow":"Friday"},{"time":"09:00"},{"conference-categories-count":{"Small":4,"Phone Booth":0,"Medium":2,"Large":2}}],[{"dow":"Friday"},{"time":"11:00"},{"conference-categories-count":{"Small":1,"Phone Booth":0,"Medium":8,"Large":3}}],[{"dow":"Tuesday"},{"time":"09:00"},{"conference-categories-count":{"Small":1,"Phone Booth":0,"Medium":2,"Large":1}}],[{"dow":"Friday"},{"time":"15:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":3,"Large":2}}],[{"dow":"Monday"},{"time":"12:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":5,"Large":3}}],[{"dow":"Monday"},{"time":"14:00"},{"conference-categories-count":{"Small":5,"Phone Booth":0,"Medium":5,"Large":2}}],[{"dow":"Thursday"},{"time":"09:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":7,"Large":3}}],[{"dow":"Monday"},{"time":"10:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":6,"Large":1}}],[{"dow":"Monday"},{"time":"16:00"},{"conference-categories-count":{"Small":7,"Phone Booth":0,"Medium":2,"Large":3}}],[{"dow":"Monday"},{"time":"15:00"},{"conference-categories-count":{"Small":6,"Phone Booth":0,"Medium":5,"Large":3}}],[{"dow":"Tuesday"},{"time":"13:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":3,"Large":0}}],[{"dow":"Tuesday"},{"time":"15:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":4,"Large":1}}],[{"dow":"Wednesday"},{"time":"11:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":2,"Large":2}}],[{"dow":"Thursday"},{"time":"14:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":8,"Large":4}}],[{"dow":"Thursday"},{"time":"16:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":3,"Large":4}}],[{"dow":"Tuesday"},{"time":"11:00"},{"conference-categories-count":{"Small":4,"Phone Booth":0,"Medium":3,"Large":1}}],[{"dow":"Friday"},{"time":"12:00"},{"conference-categories-count":{"Small":1,"Phone Booth":0,"Medium":4,"Large":2}}],[{"dow":"Monday"},{"time":"09:00"},{"conference-categories-count":{"Small":3,"Phone Booth":0,"Medium":2,"Large":1}}],[{"dow":"Wednesday"},{"time":"13:00"},{"conference-categories-count":{"Small":4,"Phone Booth":0,"Medium":4,"Large":2}}],[{"dow":"Wednesday"},{"time":"15:00"},{"conference-categories-

count":{"Small":6,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Thursday"},{"time":"10:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":8,"Large":4}}],[{"dow":"Thursday"},{"time":"12:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Friday"},{"time":"10:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":5,"Large":3}}],[{"dow":"Friday"},{"time":"16:00"},{"conference-categories-
count":{"Small":2,"Phone
Booth":0,"Medium":2,"Large":2}}],[{"dow":"Monday"},{"time":"13:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":6,"Large":2}}],[{"dow":"Friday"},{"time":"14:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Monday"},{"time":"11:00"},{"conference-categories-
count":{"Small":8,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Wednesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":2,"Phone
Booth":0,"Medium":4,"Large":1}}],[{"dow":"Tuesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":4,"Large":0}}],[{"dow":"Tuesday"},{"time":"14:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":6,"Large":1}}],[{"dow":"Tuesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":3,"Large":0}}],[{"dow":"Wednesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":3,"Large":2}}],[{"dow":"Thursday"},{"time":"15:00"},{"conference-categories-
count":{"Small":3,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Wednesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":2,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Tuesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":6,"Phone
Booth":0,"Medium":3,"Large":1}}],[{"dow":"Wednesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":4,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Friday"},{"time":"13:00"},{"conference-categories-
count":{"Small":2,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Thursday"},{"time":"13:00"},{"conference-categories-
count":{"Small":7,"Phone Booth":0,"Medium":6,"Large":4}}]]

B2_ W1.txt

[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":1,"Large":5}}],[{"dow":"Thursday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":2}}],[{"dow":"Friday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":2}}],[{"dow":"Friday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":4,"Large":4}}],[{"dow":"Tuesday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Friday"},{"time":"15:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":1,"Large":5}}],[{"dow":"Monday"},{"time":"12:00"},{"conference-categories-count":{"Small":0,"Phone Bcoth":0,"Medium":5,"Large":5}}],[{"dow":"Monday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":5,"Large":4}}],[{"dow":"Thursday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Monday"},{"time":"10:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":6,"Large":5}}],[{"dow":"Monday"},{"time":"16:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":5,"Large":5}}],[{"dow":"Monday"},{"time":"15:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":6,"Large":5}}],[{"dow":"Tuesday"},{"time":"13:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":5,"Large":5}}],[{"dow":"Tuesday"},{"time":"15:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":4,"Large":5}}],[{"dow":"Wednesday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":5,"Large":5}}],[{"dow":"Thursday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":5}}],[{"dow":"Thursday"},{"time":"16:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Tuesday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Friday"},{"time":"12:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":2}}],[{"dow":"Monday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":6,"Large":2}}],[{"dow":"Wednesday"},{"time":"13:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":5,"Large":5}}],[{"dow":"Wednesday"},{"time":"15:00"},{"conference-categories-

count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Thursday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Thursday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Friday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":1,"Large":5}}],[{"dow":"Friday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Monday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Friday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Monday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Wednesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Tuesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Tuesday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Tuesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Wednesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Thursday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Wednesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Tuesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":5}}],[{"dow":"Wednesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Friday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Thursday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone Booth":0,"Medium":6,"Large":3}}]]

B2_ W2.txt

[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Thursday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Friday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":1,"Large":3}}],[{"dow":"Friday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Tuesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":1,"Large":2}}],[{"dow":"Friday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":3}}],[{"dow":"Monday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Monday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Thursday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":2}}],[{"dow":"Monday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Monday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Monday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Tuesday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Tuesday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Wednesday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Thursday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":1,"Large":5}}],[{"dow":"Thursday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":1,"Large":2}}],[{"dow":"Tuesday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Friday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":3}}],[{"dow":"Monday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Wednesday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":5}}],[{"dow":"Wednesday"},{"time":"15:00"},{"conference-categories-

count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":3}}],[{"dow":"Thursday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Thursday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":2}}],[{"dow":"Friday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Friday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Monday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Friday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Monday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Wednesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Tuesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":1,"Large":4}}],[{"dow":"Tuesday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Tuesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":2}}],[{"dow":"Wednesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Thursday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Wednesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":1,"Large":3}}],[{"dow":"Tuesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Wednesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":2}}],[{"dow":"Friday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":5}}],[{"dow":"Thursday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone Booth":0,"Medium":3,"Large":5}}]]

B2_W3.txt

[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Thursday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":3,"Large":3}}],[{"dow":"Friday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":4,"Large":3}}],[{"dow":"Friday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Tuesday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":1,"Large":4}}],[{"dow":"Friday"},{"time":"15:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":4,"Large":3}}],[{"dow":"Monday"},{"time":"12:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":5,"Large":3}}],[{"dow":"Monday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":5,"Large":5}}],[{"dow":"Thursday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":4,"Large":2}}],[{"dow":"Monday"},{"time":"10:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":5}}],[{"dow":"Monday"},{"time":"16:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":5,"Large":4}}],[{"dow":"Monday"},{"time":"15:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":5,"Large":5}}],[{"dow":"Tuesday"},{"time":"13:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":6,"Large":4}}],[{"dow":"Tuesday"},{"time":"15:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":1,"Large":5}}],[{"dow":"Wednesday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Thursday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":3,"Large":5}}],[{"dow":"Thursday"},{"time":"16:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":2,"Large":4}}],[{"dow":"Tuesday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":3,"Large":4}}],[{"dow":"Friday"},{"time":"12:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":3,"Large":2}}],[{"dow":"Monday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":3,"Large":5}}],[{"dow":"Wednesday"},{"time":"13:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":6,"Large":4}}],[{"dow":"Wednesday"},{"time":"15:00"},{"conference-categories-

count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Thursday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Thursday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Friday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Friday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":2}}],[{"dow":"Monday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Friday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Monday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Wednesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Tuesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Tuesday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Tuesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Wednesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Thursday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Wednesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Tuesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Wednesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Friday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Thursday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone Booth":0,"Medium":3,"Large":3}}]]

B2_ W4.txt

[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Friday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Friday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Tuesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":4}}],[{"dow":"Friday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":5}}],[{"dow":"Monday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":3}}],[{"dow":"Monday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":5}}],[{"dow":"Monday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Monday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Monday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Tuesday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Tuesday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Wednesday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Tuesday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":5}}],[{"dow":"Friday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Monday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Wednesday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Wednesday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Friday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Friday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Monday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Friday"},{"time":"14:00"},{"conference-categories-

count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Monday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Wednesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Tuesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Tuesday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Tuesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Wednesday"},{"time":"10:00"},{"conference-categories-
coun*":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Wednesday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Tuesday"},{"time":"10:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Wednesday"},{"time":"16:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":4}}],[{"dow":"Friday"},{"time":"13:00"},{"conference-categories-
count":{"Small":0,"Phone Booth":0,"Medium":4,"Large":5}}]]

B2_ W5.txt

[[{"dow":"Wednesday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":5}}],[{"dow":"Thursday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":3}}],[{"dow":"Friday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Boot":0,"Medium":4,"Large":3}}],[{"dow":"Friday"},{"time":"11:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":5}}],[{"dow":"Tuesday"},{"time":"09:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Friday"},{"time":"15:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Monday"},{"time":"12:00"},{"conference-categories-
count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":3}}],[{"dow":"Monday"},{"time":"14:00"},{"conference-categories-
count":{"Small":0,"Phone

Booth":0,"Medium":6,"Large":4}}],[{"dow":"Thursday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Monday"},{"time":"10:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":2,"Large":5}}],[{"dow":"Monday"},{"time":"16:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":1,"Large":4}}],[{"dow":"Monday"},{"time":"15:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":3}}],[{"dow":"Tuesday"},{"time":"13:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Tuesday"},{"time":"15:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":2}}],[{"dow":"Wednesday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Thursday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Thursday"},{"time":"16:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Tuesday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":4}}],[{"dow":"Friday"},{"time":"12:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Monday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Wednesday"},{"time":"13:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Wednesday"},{"time":"15:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Thursday"},{"time":"10:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Thursday"},{"time":"12:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Friday"},{"time":"10:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Friday"},{"time":"16:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":3}}],[{"dow":"Monday"},{"time":"13:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":3}}],[{"dow":"Friday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Monday"},{"time":"11:00"},{"conference-categories-count":{"Small":0,"Phone

Booth":0,"Medium":3,"Large":4}}],[{"dow":"Wednesday"},{"time":"09:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Tuesday"},{"time":"16:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":4}}],[{"dow":"Tuesday"},{"time":"14:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":4,"Large":5}}],[{"dow":"Tuesday"},{"time":"12:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":1,"Large":3}}],[{"dow":"Wednesday"},{"time":"10:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":5,"Large":5}}],[{"dow":"Thursday"},{"time":"15:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Wednesday"},{"time":"12:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":1,"Large":2}}],[{"dow":"Tuesday"},{"time":"10:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":4}}],[{"dow":"Wednesday"},{"time":"16:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":3,"Large":5}}],[{"dow":"Friday"},{"time":"13:00"},{"conference-categories-count":{"Small":0,"Phone
Booth":0,"Medium":6,"Large":5}}],[{"dow":"Thursday"},{"time":"13:00"},{"conference-categories-count":{"Small":0,"Phone Booth":0,"Medium":5,"Large":3}}]]

PYTHON

BY

Mr. LOKESH sir

LOKESH IT

Flat: 302, Sree Swati Anukar, Beside: Aditya Trade Centre,
Ameerpet, Hyderabad - 16.
Ph: + 91 90307 49297, +91 80740 96308