

### Why do learn code?

Programming is a powerful tool you can use to solve all kinds of problems by using code.

### What do you want to do?

- ❖ Build a phone app to help you find directions.
- ❖ See what people are saying about your business on social media.
- ❖ Calculate the how much money you want to buy a car.

### What is python?

Python is a interpreter, interactive, object-oriented & high level programming language and scripting language also .it was developed by **Guido Van Rossum** in 1985-1990, it was developed mathematical & science Research center in Netherland, it's available under the general public license.

### what is the overview of python?

- ❖ Scripting language
- ❖ Object-oriented
- ❖ Interpreter
- ❖ Interactive
- ❖ Easy to learn
- ❖ Contains good features from other programming language

### What is the features of python?

- ❖ Easy to learn
- ❖ Easy to read

- ❖ Huge libraries
- ❖ Portability
- ❖ Interface to all major databases
- ❖ GUI programming









Etc.

### What applications are built by using python?

- ❖ Web applications
- ❖ Games applications
- ❖ Desktop applications
- ❖ Database programming applications
- ❖ Task Scheduling
- ❖ Data Analysis
- ❖ Image processing
- ❖ Artificial intelligence

Etc.

### Where we are using python?

-  YouTube
-  Google
-  Bittorent
-  Quora
-  Yahoo maps
-  Spotify
-  NASA
-  Reddit

Etc

## Why we are choosing python?

Remain those all programming languages are compiled based programming languages.

- Python is a interpreted programming language
- We do not need to write data types
- less coding
- Scripting language
- less time taken process compare then other programming languages
- Multiple inheritance is directly we do in python
- General English language

## Environment Setup:

- Download the required python 2.7.14 Software from the following Website  
<https://www.python.org/downloads>
- Click on the downloaded installer file.
- Click on the run.
- Click on the next.
- Click on the next.
- Click on the next.
- Click on yes.
- Click on finish.

## Development of python applications (or) programs:

We can develop python applications/programs in 2 modes

1. Interactive mode
2. Batch mode

- Interactive mode is a command line shell.
- In command line shell, if we write any python statements immediately execute and show the result.
- Interactive mode is used to test the features of the python.

**NOTE:** interactive mode is not used for development of the business applications.

**IDLE mean integrated development environment**

## 2. Batch Mode:

- In batch mode we write group of python statements in any one of the editors (or) IDE's
- Different editors are
- Notepad, Notepad++, Edit plus, IDLE

Different IDE's are

- PyCharm, Eclipse, Eric
- After writing the group of python statements in anyone of the editor/IDE we save that file with extension “.py” or “.pyw”
- Batch mode is used for mainly business development applications.

## Scripting language

## programming language

1. Scripting language is interpreter 1. Programming language is

Based languages

compiler based languages

2. Scripting language programs

2. It requires explicit compilation

Compilation is not required

3. s.l.p applications directly

3. p.L applications we cannot run

We can run

without compiling

4. Scripting language programs

4. P.L applications take less time to

Takes longer time to execute

execute.

Ex: shell script, perl...

Ex: c, c++, java, .Net

**Note:** most of the people calling python is a scripting language because the way of developing python applications and execution of python applications are similar to the scripting languages.

### **Features of python:**

#### 1. The syntax of the python languages is very simple.

- Anybody can remember the python language syntax's, rules and regulations very easily.
- By developing the python applications and programs are need not to focus on syntax's
- Without having any other programming language knowledge directly anybody can learn python language.

#### 2. Free and open source:

- Anybody can use the python software without purchasing license agreement of python.
- Anybody can read the python source code & they can do the modifications in the python source code and we can redistribute that code to others.

### 3. High-level language:

While developing the python programs need not to focus on the memory management and memory used by the program (low level details)

### 4. Python supports oop's:

Python is language supports p.o.p & o.o.p language features

(Procedure oriented programming(c) & (object oriented programming)

Different oops principles are

- Encapsulation
- Polymorphism
- Inheritance
- Abstraction

### 5. Interpreter language:

- Python application does not require explicit compilation, so that compiler is not required for the python software.
- Directly we can run the python program applications without compiling
- Python interpreter is responsible for the execution of the python applications
- Whenever we run the python code, python interpreter will check the syntax error. If no syntax error python interpreter converts that code in the form of intermediate code ( low level form and executes it)
- The intermediate code of the python applications is known as byte code.

The extension of the byte code file is .pyc (python compile file)

## 6. Extensible:

- Python application execution is slower compared to C, C++ programs.
- To overcome the above problem we can implement some logics by using c, c++ language features and we can use c, c++ programs into application.
- Python source code does not contain security i.e anybody can read python code and they can do modifications in the source code.
- We have some code in java (like 10000 lines of code), we are not re write that java code in python again, just we can use that 10000 code in python directly, it is possible by using NOTATION method are done. Non-python code also allows implementing in python.

## 7. Embeddable:

- We can embeddable the python code into the other language programs such as c, c++, java and .Net.....
- In order to provide the scripting capabilities to the other language programs/applications, we use python code into those applications.
- Python have huge building libraries.
- By using built-in libraries application development will become faster.

## Python Memory Management

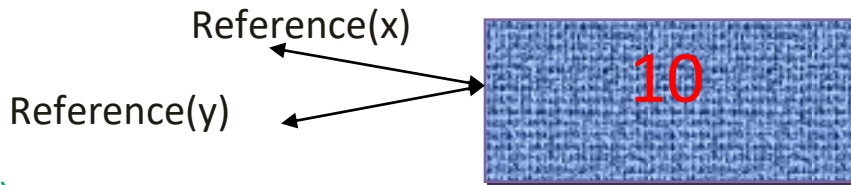
```
x = 10
```

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

```
y = x
```

```
if (id(x)==id(y)):
```

```
    print ("x and y refer to the same object")
```



**Note:** everything is object in python

```
x = x+1
```

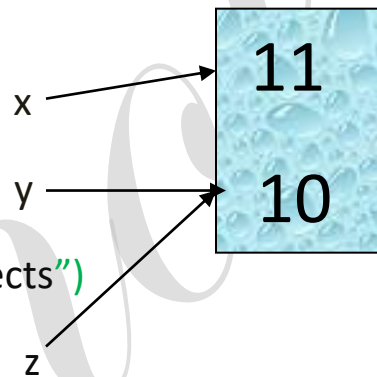
```
if (id(x) != id(y)):
```

```
    print("x and y refers different objects")
```

```
z = 10
```

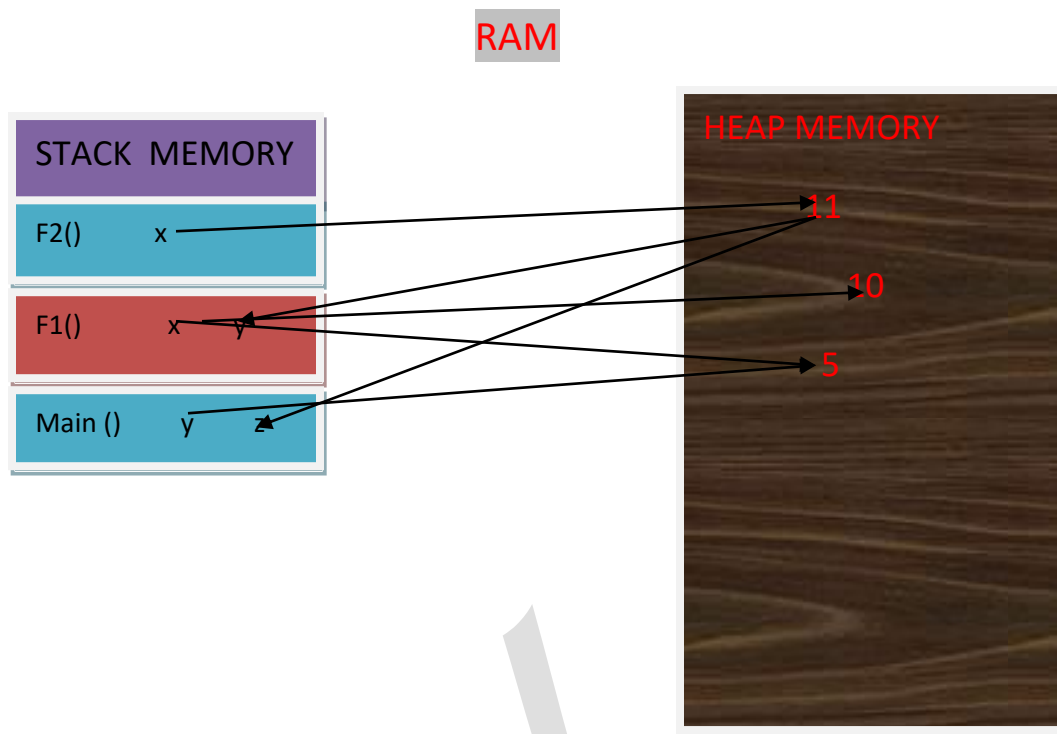
```
if (id(y)==id(z)):
```

```
    print("y and z reference both are same memory")
```



**Note:** Alternately change the data, that's what means to say python dynamically typed language (if you want to change the **z** data again it will create the new object check the above examples)





Os, other process, applications and shared memory

**Note:** The methods are executed from stack memory by default, the Program begins to execute under main method stack.

- ✚ The objects created in HEAP Memory where as references created in STACK Memory.

Program for above diagram flow

```
def f2(x):
```

```
    x = x+1
```

```
    return x
```

```
def f1(x):
```

```
    x = x * 2
```

```
y = f2(x)
```

```
return y
```

```
# main
```

```
Y = 5
```

```
Z = f1(y)
```

**Note:** as per as above program finally we have only main () block (stack) and 5, 11 values are in Heap, so alternatively changes by default of that reason python have dynamic memory, check the above diagram.

Ex:

Class Car:

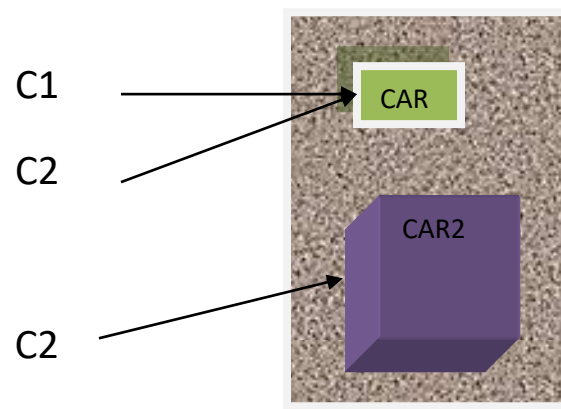
```
def __init__(self, w):  
    self.wheels = w  
  
def getWheels (self):  
    return self.wheels
```

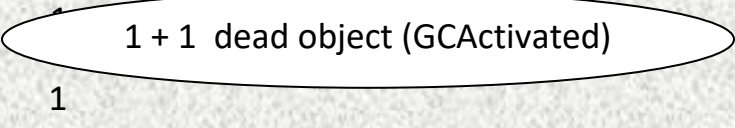
```
C1 = Car (4)
```

```
C2 = C1
```

```
C1 = None
```

```
C2 = Car (4)
```



OBJECT	Number of References
CAR OBJECT 1	
CAR OBJECT 2	

The above diagram, newly created object have one reference c1 and again share the same object c2 both have same object, automatically increase the reference count ( 1+1), let's now remove reference from c1 count will be decrease, let us assign the new object to c2 then new row added to the table (object2), the car object1 becomes zero that deleted such has no reference these are called dead objects, python has one mechanism that is called Garbage Collector, automatically it's removes the data object from memory permanently.

### Difference memory allocations in programming languages

Statement	Python X = 10	Java Int x = 10
Data Type declaration	Not Needed. Dynamically typed	A primitive data type stored in 4 bytes
What is 10?	An object created on Heap Memory	Memory location where 10 is stored
What does x contain?  X = x + 1	Reference to object10  X starts new reference to new object whose value is 11	Memory location where 10 is stored  X continues to point to the same memory, with

		value equal to 11
X = 10 Y = 10	Both x and y will refer to the same object.	X and y variables pointing to different memory locations.

## PYTHON

- + The python methods and variables created on stack memory.
- + The object and instance variables are created on Heap Memory.
- + A new stack frame is created on invocation of function/method.
- + Stack frames are destroyed as soon as the function/method returns.
- + Garbage collector is a mechanism to clean up the dead objects.

### **Two types of comments in python:**

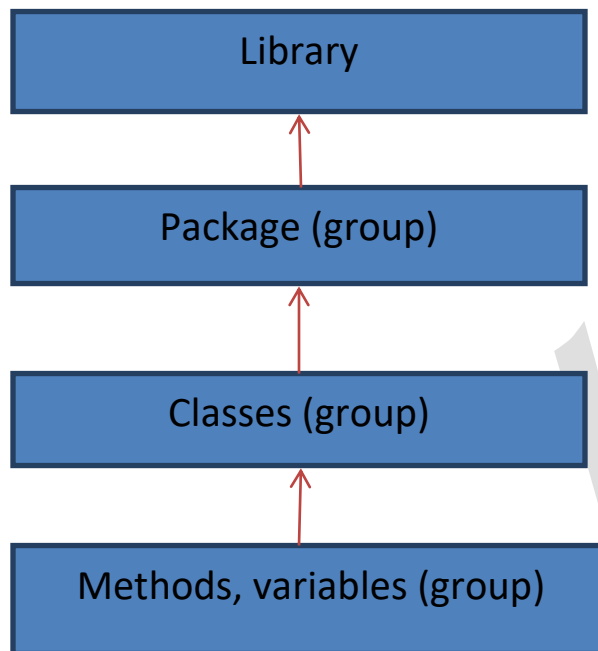
#### 1. Single-line comment

These comments start with '#'

#### 2. Multiline comments

These comments are start with '''hello python''' (or) """hello Guido van Rossum"""

# This bar chart for the combination of Library features



### What is variable?

**Python - Variables** are nothing but reserved memory locations to store values. This means that when you create a **variable** you reserve some space in memory. Based on the data type of a **variable**, the interpreter allocates memory and decides what can be stored in the reserved memory.

- Always variables names are start with character
- The variables are use to store the data temporary

a = 1, number = 2, number\_one = "hello", number\_123 = 4


Ex: a = 10, var = "ashok"

### What is parameter?

A parameter is a variable to receive the data into the method.


## What is an arguments?

The data passed to method is called arguments.

Ex: `def sum(a,b):` parameters 

`Print " welcome to python world "`

`return a+b`

`sum(10,20)` arguments 

**Data types:** a data type represents the type of data stored in a variable or (memory)

### **Fundamental data-types:**

- The variables which are represented with fundamental data types store the address of the object in which we can represent only one element

Ex: int, float, long, complex, bool, str

### **Built-in functions in python:**

Program: python supports so many built-in functions.

`type()`:

- ❖ This function is used to know the data-type of the variable.

`Id()`:

- ❖ This function is used to know the address of the object which is pointed by the variables.

Print():

- ❖ This function is used to display the data and pass the cursor to the next point.

'int': These data types represents number without a decimal point

Ex: i = 414

Print (i)

type(i)

id(i)

'long'

i = 414123456983645

Print (i)

type(i)

id(i)

'float': These data types are represent the number with decimal point

a = 143.4

print(a)

type(a)

id(a)

'complex': Mathematical operations for complex numbers

a = 3+4j

print(a)

type(a)

id(a)

'bool': These data types are represent the boolean like True (or) False

a = False (or) True

print (a)

type(a)

id(a)

'str': A string represents a group of characters combined one form

s = "python"

print(s)

type(s)

id(s)

### **Reading the data from keyboard:**

❖ We can read the data from keyboard by using following functions

1. input()    2. raw\_input()

1. input(): It is used to read the data from the keyboard directly in our required format.

2. raw\_input(): It is used to read the data from the keyboard in the form of string .



### Type conversion functions:

- ❖ Type conversions are used to convert one data type into another data type

Q. Write a python program to perform addition of two numbers by reading those numbers from keyboard using `raw_input()`, `input()`

```
X=raw_input('enter the number')
```

```
Y=raw_input('enter the number')
```

```
print type(x)
```

```
print type(y)
```

```
print x+y
```

```
z = int(x)
```

```
print type(z)
```

```
print z
```

```
m = int(y)
```

```
print type(m)
```

```
print m
```

```
print z+m
```

2. `long()`:

- ❖ It is used to convert the string representation data in the form of long data type format

```
X=raw_input("enter long number")
```

Print type(x)

Print x

Z=long(x)

Print type(z)

Print z

3.float():

- ❖ It is used to convert the string represent data in the form of float data type format.

X=raw\_input("enter float number")

Print type(x)

Print x

Y=float(x)

Print type(y)

Print (y)

4. complex (): It is used to convert the complex representation data in the form of string data type format.

X=input("please enter complex number")

Print type(x)

Print x

Z=str(x)

Print type(z)

Print z

5. bool(): It is used to convert the string represented data in the form of bool data type format.

X=raw\_input("enter the bool data")

Print type(x)

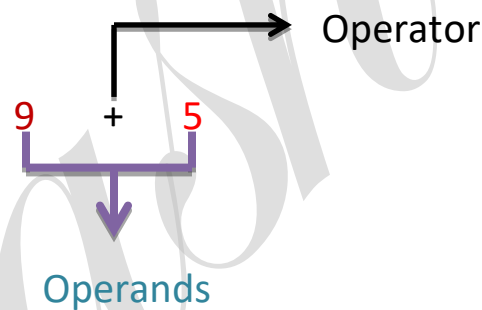
Print x

Y=bool(x)

Print type(y)

Print y

**Operators:** It is used to manipulate the values of the operands.



Python supports the following types of operations.

1. Arithmetic operations
2. Comparison (relational) operators
3. Logical operators
4. Bitwise operators

## 5. Assignment operators

## 6. Special operators

**1. Arithmetic operations:** These operations are used to perform mathematical operations between two variables.

Operator	Meaning
+	add two operands (or) unary plus
-	Subtract right operand from the left (or) unary Minus
*	multiply two operands
/	divide left operand by right one
%	modules – remainder of the division of left Of operand by right
**	Exponent – left operand raised to the power of right
//	floor division – division that results into whole number adjusted to the left in number line

Ex: 1

x = 10

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)
```

Ex: 2

```
x = "ashok"
y = 2
print('x*y = ', x*y)
print('y*x = ', y*x)
```

**2. Comparison/Relational operations:** These operations are used to compare the values.

- Comparison operators return either True/False

**Operator**

**Meaning**

>	greater than – True if left operator is greater Than right one
<	less than – True if both operand is less than right one
==	equal to -> True if both operands are equal
!=	not equal to -> True if both not equal

`>=`

greater than (or) equal to -> True if left operand is greater than or equal to right

`<=`

less than (or) equal to -> True if left operand is less than or equal to right

Ex: 1

`x = 4`

`y = 6`

`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)`

Ex: 2

`x = "python"`

`y = "guido"`

`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)
```

3. **Logical operators:** it is used to perform logical operator.

- logical operators written either True/False values.

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

**Operator**

**Meaning**

AND

True if both operands are True

OR

True if either of the operand is True

NOT

True if operand is false

Ex:

```
x = True
```

```
y = False
```

```
print('x and y is', x and y)
```

```
print('x or y is', x or y)
```

```
print('x not y is', not x)
```

4. **Bitwise operator:** These operations are used to perform the operations on the value.

Operator	Meaning
&	Bitwise AND
~	Bitwise NOT
	Bitwise OR
^	Bitwise XOR
<<	Bitwise left shift
>>	Bitwise right shift

Ex:

```
x = 10
```

```
y = 4
```

```
print("x&y is:",x&y)
```

```
print("x|y is:",x|y)
```

```
print("~x is:",~x)
```

```
print("x^y is:",x^y)
```

```
print("x>>y is:",x>>y)
```

```
print("x<<y is:",x<<y)
```

**NOTE:** Bitwise operators internally converts the operands values in the form of binary format and performs the operation and gives the results in the form of decimal format.



**5. Assignment operations:** These operations are used to assign the values to the variables.

- it contains both bitwise & arithmetic operations.

operator	example	Equivalent to
=	x = 4	x = 4
+ =	x+ = 4	x = x+4
- =	x - = 5	x = x-5
* =	x* = 2	x = x*2
/ =	x/ = 2	x = x/2
% =	x% = 2	x = x%2
// =	x// = 4	x=x%4
** =	x** = 4	x=x**4
& =	x& = 4	x=x&4
=	x  = 4	x=x 4
^ =	x^ = 4	x=x^4
>> =	x>> = 4	x=x>>4
<< =	x<< = 4	x=x<<4

Ex:

```
x=10
```

```
print("x=10:",x)
```

```
x+=5
```

```
print ("x+=5:",x)
```

```
x-=5
```

```
print ("x-=5:",x)
```

`x*=5`

`print ("x*=5:",x)`

`x**=5`

`print ("x**=5:",x)`

`x/=5`

`print ("x/=5:",x)`

`x%=5`

`print ("x%=5:",x)`

`x//=5`

`print ("x//=5:",x)`

`y=10`

`y&=5`

`print ("x&=5:",y)`

`y^=5`

`print ("x^=5:",y)`

`y>>=5`

`print ("x>>=5:",y)`

`y<<=5`

`print ("x<<=5:",y)`

## 6. Special operations:

Python supports the two types of special operators.

1. Identity operators
2. Membership operators

**1. Identity operators:-** These operators are used to compare the addresses of the memory locations which are pointed by the operands.

- Identity operators returns true (or) false

Operator	Meaning
is	True if the operands are identical
is not	True if the operands are not identical

Ex:

```
x=10
```

```
y=10
```

```
x1="bye"
```

```
y1="bye"
```

```
print("x is y:",x is y)
```

```
print("x is not y:",x is not y)
```

```
print("x1 is y1:",x1 is y1)
```

```
print("x1 is not y1:",x1 is not y1)
```

**2. Membership operators:-** These operators are used to search for a particular element in a string like list, tuple, set.....

- Membership operators returns True or False value

## Operator

## Meaning

in	True if the value/variable is found in sequence
not in	True if the value/variable is not found in sequence

Ex:

```
x="well come to python"
```

```
print('h' in x)
```

```
print('to' not in x)
```

```
print('come' in x)
```

```
print('python' not in x)
```

### Decision Making:

#### Conditional statements:

- ❖ These statements are used to decide whether code has to be execute (or) skip based on the given condition.
- ❖ After executing the condition, it should be return either True or False.

Elements of Conditional statements: It support two types of elements

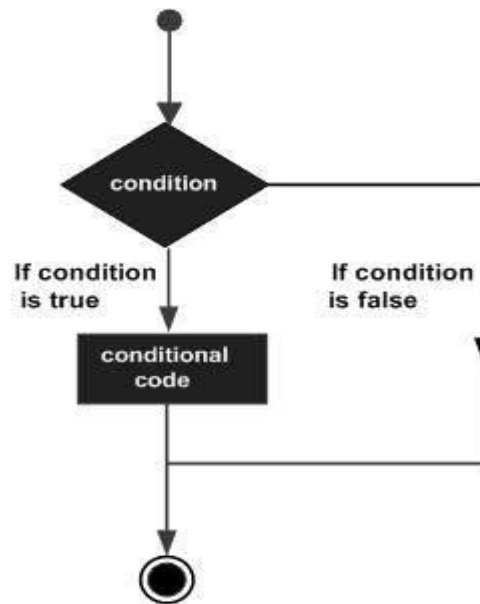
1. Condition

2. Block

1. Condition:- Any expression which returns either True or False value after evaluating that expression is known as “ condition ”

2. Block:- All the statements are following same space indentation is known as “ Block “

- Blocks begin with when the indentation increases.
- Blocks can contain other blocks.



Python supports three conditional statements, these are

1. if (or) simple if
2. if else
3. elif

1. if (or) simple if:- It is used to check the a condition, if the condition satisfies it will enter into the block of statements, if the condition fail it will exit (skip)

syntax:-            **if condition: statement**

**(or)**

**If condition:**

**Statement 1**

## Statement 2...

Ex: 1

```
a=10
```

```
b=4
```

```
if a>=b:
```

```
    print 'a is bigger'
```

```
    print 'a may be equal to b'
```

Ex: 2

```
name = raw_input("enter any name")
```

```
if name == "python":
```

```
    print "hello wellcome"
```

```
    print("what about you")
```

2.if - else:- In if-else condition returns True then it will execute if block otherwise it will execute else block.

Syntax: **if condition:**

**statement 1**

**statement 2**

**else:**

**statement 1**

**statement 2**

Ex:1

a=10

b=6

if a>b:

    print "true"

else:

    print "false"

Ex:2

name = input("enter your name")

if name == "ashok":

    print "hello ashok well come"

else:

    print "it's not match"

3.elif :- We will use else-if ladder to work with multiple conditional statements.

Syntax: **if condition:**

**Stmnt 1**

**Stmnt 2**

**elif condition:**

**stmnt 1**

stmnt 2

(optional)else:

Stmnt 1..

Ex:1

```
name=raw_input("please enter your name:")
```

```
age=input("enter your age")
```

```
if age<12:
```

```
    printname,"your are kid"
```

```
elif age<19:
```

```
    print name,"your in teenage"
```

```
elif age<40:
```

```
    print name,"your young"
```

```
else:
```

```
    print name,"old"
```

Ex:2

```
a=10
```

```
b=15
```

```
if a>b:
```

```
    print" a is big"
```

```
elif a<b:
```



```
print" b is big"
```

```
else:
```

```
print"a and b are equal"
```

task1:- write a program to find biggest number among 2 numbers.

task2:- write a program to find biggest number among 3 numbers.

### Loops:

Loops are used to execute the block of statements repeatedly as long as given condition satisfy.

### Python supports two types of looping statements:

1. for loop

2. While loop

1. for loop: For loop executes the block of statements (or) set of statements with respect to every element of the given string.

Ex

```
sum = 0
```

```
for i in [1,2,3,4]:
```

```
    sum = sum+i
```

```
print sum
```

range: range () generates the group of values based on the given range and gives it as a list.

- Python has a built-in function called range that can easily generate a range of whole numbers, there is another built-in function call “xrange” that provides the same result, but uses of lot less memory.

Ex

```
a = 10
```

```
print range(a)
```

```
print range(1,11)
```

```
print range(0,30,5)
```

Note: xrange () also as it is range

Ex

```
for x in range(10):
```

```
    print x
```

```
for y in range(0,10,2):
```

```
    print y
```

```
for z in range(10):
```

```
    if z%2 == 0:
```

```
        print z
```

**Task1: Write a program to print even numbers for given list  
[4,8,5,12,2,0,6,7,9,11]**

2. while loop: While loop executes the set of statements or block repeatedly until condition will become false.

Ex-1

```
name = ' guido'

while name != 'python':

    name = raw_input('enter name')

print "thank you"
```

Ex-2

```
sum = 0

spam = 1

while spam<11:

    sum = sum+spam

    spam = spam+1

print sum
```

Infinite Loop:

Ex

```
while True:

    name = raw_input("please enter any name:")

print name
```

## Control Statements

- The statements which change the flow of execution are called control statements.
- Control statements are used for to write better and complex programs.

Break: Break statements used to exit from the loops.

- Generally we use the control statement in infinite loops.

Ex

while True:

```
    name = raw_input("enter any name:")
```

```
    if name == 'python':
```

```
        break
```

```
print "thank you"
```

Continue: Continue statement can be used in the loops.

- Continue statement is used in a loop to execute next iteration of the loop, when continue is executed subsequent statements in the loop are not executed.

Ex

while True:

```
    name = raw_input("enter any name:")
```

```
    if name != "python":
```

```
    continue

print "hello ashok, please enter the password"

password = raw_input("enter the password")

if password == "python":

    break

print "access granted"
```

Pass: Syntactically, block is required but we don't want to perform any action logically, then we can replace the block with 'pass'.

- Pass is a keyword it does nothing.

Ex

```
i = 10

while i<=10:

    pass

if i<5:

    pass

for i in range(20):

    pass
```

### **Data structures:**

Group of elements enclosed by any particular symbol is known as data structure. Every data structure should contain one major memory

allocation & elements in the data structure should share the major locations into parts.

Data structures are

- List
- Tuple
- Dictionary
- Set
- String
- Number

Mutable and immutable objects:

Mutable: if we are able to alter (CRUD) the given data structure that is called mutable objects.

Immutable: if we are not able to alter (CRUD) given data structure that is called immutable objects.

Mutable objects

Immutable objects

**List**

**Tuple**

**Dictionary**

**String**

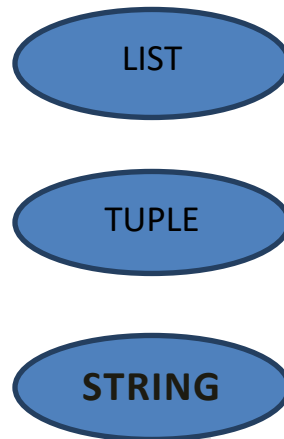
**Set**

**Number**

Sequence:

In python **sequence** is the generic term for an ordered set, we can retrieve the any value in sequence order.

### Sequence Data types



Ex:

-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
A	S	H	O	K	R	E	D	D	Y
0	1	2	3	4	5	6	7	8	9

The above example is following sequence order list, tuple, string

**List:** list is used to represent the group of elements are in single entity (or) object.

- it is represented by square brackets " [ ] "
- it is mutable
- insertion order is preserved
- duplicate elements are allowed
- every element in the list object represented with unique index
- list supports the both positive and negative indexes

Ex: l = [1,2,3,4]

Print l

**Tuple:** It is used to represent group of elements into a single entity

- tuple objects are immutable objects i.e, once if you create any tuple object later you cannot change.
- it is represented by “ ( ) ”
- it is following sequence order
- duplicate elements are allowed
- it supports the both forward and backward indexes
- fixed data purpose we are using tuple
- Performance is fast compare then list

Ex: a= (“guido”, “python”)

Print a

**String:** The group of characters is formed one form is called string

- to represent the string in python we use ‘str’ data type
- insertion order is preserved
- it is immutable
- we can the string object in python 2 ways those are

1. by using ‘ ’ we can represent the single line string

2. by using “ ” we can represent the multiple lines string

- every character in string object have unique index
- string also supports the both forward and backward indexes

Ex: x = “python”



Print x

```
X = '''hello python
```

```
Developers welcome
```

```
To new world'''
```

Print x

Sequences ( list, tuple, string)

- all these examples applicable to list, tuple, string

**indexing:** It is used to access any item in the sequence using it's index

List: Ex:

```
l = 'python'
```

```
print(l[1]) # y
```

tuple: Ex:

```
t = 'guido'
```

```
print(t[3]) # d
```

string: Ex:

```
s = "ashok"
```

```
print(s[-5]) # a
```

**slicing:** x[1:4]

- slice out sub-strings, sub-lists, sub-tuples using indexes  
[start:end+1]

List: Ex

```
x = [1,2,3,4]
```

```
print x[1:3]    # [2,3]
```

Tuple: Ex

```
x = (9,8,7,6,5)
```

```
print x[-4:-1]  # (8,7,6)
```

String: Ex

```
y = "johns"
```

```
print y[2:4]    # hn
```

**Adding/concatenating:** It is used to combine two sequences of the same type using addition(+)

List: Ex

```
x = ['pig', 'cow'] + ['dog']
```

```
print x    # ['pig', 'cow', 'dog']
```

**Multiplying:** It is used to multiply a sequence using ' \* '

String: Ex

```
s = "bug"*3
```

```
print(s)    # 'bugbugbug'
```

List: Ex

```
l = [1,4]*2
```

```
print(l) # [1,4,1,4]
```

```
t=("ashok")*2
```

```
print(t) # ashokashok
```

**checking membership:** It is used to test whether an item is in (or) not in a sequence.

String: Ex

```
s = "coding"
```

```
Print('d' in s) # True
```

List: Ex

```
l = ["srinu", "bhaskar", "ashok"]
```

```
print("ashok" not in l) # False
```

Tuple: Ex

```
t = ("jhon", "ivanka", "obama")
```

```
print("obama" in t) # True
```

**iterating:** It is used to iterate through items in a sequence

item

list: Ex

```
x = [1,2,3]
```

```
for item in x:
```

```
    print(item*2) # 2,4,6
```

### index & item

for index,item in enumerate(x):

print(index,item) #(0,1)(1,2)(2,3)

### tuple: Ex

x = ['a','b','c']

for item in x:

print(item\*2)# aa,bb,cc

for index,item in enumerate(x):

print(index,item) #(0,a)(1,b)(2,c)

### string: Ex

x = "ashok"

for item in x:

print(item\*2) # aa,ss,hh,oo,kk

for index,item in enumerate(x):

print(index,item) #(0,a)(1,s)(2,h)(3,o)(4,k)

**number of items:** It is used to count the member of items in sequence

### list: Ex

l = [1,2,3,4]

print(len(l)) # 4

### string: Ex

```
s="swift"
```

```
print(len(s)) # 5
```

tuple: Ex

```
t="R"
```

```
print(len(t)) # 1
```

**Minimum:** It used to find the minimum item in a sequence

- alpha or numeric types but can't mix types

List: Ex

```
X = ['pig', 'cow', 'horse']
```

```
Print (min(x))
```

String: Ex

```
X = 'bug'
```

```
Print (min(x))
```

Tuple: Ex

```
x = ("java","python","Net")
```

```
print(min(x))
```

**Maximum:** Find the maximum item in a sequence

List: Ex

```
x = ["java","python","Net"]
```

```
print(max(x))
```

### String: Ex

```
x = "python"  
print(max(x))
```

### Tuple: Ex

```
x = ("java","python","Net")  
print(max(x))
```

**Sum:** Find the sum of items in a sequence

- entire sequence must be numeric type

### List: Ex

```
x = [1,2,3,4]  
print(sum(x))
```

- String not possible because of all numeric values for these operations, if you write any program get error.

### Tuple:Ex

```
x=(1,2,3)  
print(sum(x))
```

**Sorting:** Returns a new list of items in sorted order

- doesn't change the original list

### List:Ex

```
x = ['hyd','bengulore','chenai']
```

```
print(sorted(x))
```

#### String:Ex

```
x = "python"
```

```
print(sorted(x))
```

```
x = "guido","games"
```

```
print(sorted(x))
```

#### Tuple: Ex

```
x = ("x","a","y")
```

```
print(sorted(x))
```

**Count:** Returns count of an items

#### List:Ex

```
x = ['a','a','b','c']
```

```
print(x.count('a'))
```

#### String:Ex

```
s = "hippo"
```

```
print(s.count('p'))
```

#### Tuple:Ex

```
x = ('python','java','python')
```

```
print(x.count('python'))
```

**Index:** It is used to returns the index of the first occurrence of item.

### List:Ex

```
x = ['pig','cow','pig']  
print(x.index('cow'))
```

### String:Ex

```
x = 'pythony'  
print(x.index('y'))
```

### Tuple:Ex

```
x = ('a','b','b')  
print(x.index('b'))
```

## List

- List is used to represent the group of elements into a single object
- Duplicate elements are allowed
- Insertion order is preserved
- Every element in the list object have unique index
- List supports the both positive and negative index
- List objects are mutable
- List represented by “[ ]”

Delete: Delete a list or item from a list

```
l = [9,8,6,7]  
del(l[0])
```



```
print(l)
```

```
del(l)
```

```
print(l)
```

Append: This method is used to insert one element at the end of the list

- The element can be of any type

Ex

```
l = [1,2,3,4]
```

```
l.append(10)
```

```
print l
```

Extend: This method is used to insert multiple elements at the end of the list by passing a data-structure as input

Ex

```
l = [1,2,3]
```

```
l1 = [4,5,6]
```

```
l.extend(l1)
```

```
print l
```

```
l = [1,2,3,4]
```

```
l.extend([5])
```

```
print l
```

Insert: This method is used to insert one element at any position in a list

Ex

```
l = [1,2,3]
```

```
l.insert(2,'python')
```

```
print l
```

POP: This method is used to remove the last element from a list

```
l = [1,2,3]
```

```
l.pop()
```

```
print l
```

Remove: This method is used to remove first instance of an item

```
l = ['a','b','c']
```

```
l.remove('a')
```

```
print l
```

Reverse: This method is used to reverse the element in a list

- This method doesn't require any parameter

```
l = ['a','b','c']
```

```
l.reverse()
```

```
print l
```

Sort: This method is used to arrange list of elements in an ascending order

```
l = ['m','z','c']
```

```
l.sort()
```

```
print l
```

List Comprehension: Creating the list object writing business logic is known as “list comprehension”

Ex

```
a = [ x for x in range(10)]
```

```
print a
```

```
x = [ x for x in range(0,10,2)]
```

```
print x
```

```
s = [ p for p in a if p%2==0]
```

```
print s
```

### Tuple

- tuple is used to represent the group of elements into a single object
- Duplicate elements are allowed
- Insertion order is preserved
- Every element in the tuple object has a unique index
- tuple supports both positive and negative index
- tuple objects are immutable
- List represented by “( )” or optional

Constructor: Creating a new tuple

```
l = [1,2,3]
```

```
t=tuple(l)
```

```
print t
```

Immutable: Member objects may be mutable

```
x = (1,2,3)
```

```
del(x[1])    # error
```

```
x = ([1,2],8)
```

```
del(x[0][1])
```

```
print x
```

### Differences between list and tuple

List	tuple
List objects are mutable	Tuple objects are immutuable
Iterating the list is slower	Iterating the tuple is faster compare than list
By using del we can delete the elements and entire object	By using del, we can delete only entire object not an elements
If the data changes frequently, then it is recommended to represent that data by using list	Fixed data purpose we are using the tuple

## String

- The group of characters are in one form is known as string
- To represent the string in python we can use 'str' data type
- We can create the string object in python in 2 ways they are
  1. By using ' ' # we can represent only one line string
  2. By using ''' ''' # we can represent multiple lines dtring
- String objects are immutable
- Every element in the string object have unique index
- Python supports the both forward and backward indexes

Capitalize: This method is used to capitalize first character of given string.

- the output format is always a string
- this method doesn't require any parameter

Ex

```
s = "hello python"
```

```
x = s.capitalize()
```

```
print x
```

Upper: This method is used to capitalize complete given string

Ex

```
s = "bengulore"
```

```
x = s.upper()
```

```
print x
```

Lower: This method is used to lower the complete given string

Ex

```
s = 'PYTHON IS A LANGUAGE'
```

```
x = s.lower()
```

```
print x
```

Find: Find method is also used to get the index position of given string,. if the given substring is not available then it will return -1 instead of error. So, execution will not be stop any more

Ex

```
s = 'python'
```

```
z = s.find('x')
```

```
print z
```

Split: This method is used to divide the given string into multiple parts

- The output of split operation is always a list

Ex

```
s = "hi h@llo w@orld"
```

```
x = s.split()
```

```
print x
```

```
s = "hi h@llo w@orld"
```

```
x = s.split('@')
```

print x

isupper: This method is used to check whether the given string contains only the upper case characters.

- No parameters required.
- The out put format will be Boolean value.

Ex

```
s = "America"
```

```
x = s.isupper()
```

print x

islower: This method is used to check whether the given string contains only the lower case characters.

- No parameters required.
- The out put format will be Boolean value.

Ex

```
s = "dubai"
```

```
x = s.islower()
```

print x

isdigit: This method will return True, if the given string contains only numbers(0-9) as else false.

Ex

```
s = "123"
```

```
x = s.isdigit()
```

```
print x
```

isalpha: This method will return True only if the given string contains alphabetic(a-z, A-Z).

Ex

```
s = "hi"
```

```
x = s.isalpha()
```

```
print x
```

Encode: This method is used to provide encryption process on given string. This method accepts exactly one parameter.

```
s = "python"
```

```
x = s.encode("base64")
```

```
print x
```

decode: This method is used to provide de-cryption process on given string. This method accepts exactly one parameter.

Ex

```
s = "python"
```

```
x = s.encode('base64')
```

```
print x
```

```
k = x.decode("base64")
```

```
print k
```



Replace: This method is used to replace some set of string with another string. This method accepts two parameters. Parameter one indicates string to replace. Parameter two indicates string to replace.

Ex

```
s = "python"
```

```
x = s.replace('o','@').replace('y','!')
```

```
print x
```

## Set

- Set is used to represent group of immutable elements into a single object.
- These objects are mutable.
- It's not following any order.
- Duplicate elements are not allowed.

Constructor: Creating a new set

Ex

```
l = {2,3,2,3}
```

```
x = set(l)
```

```
print x
```

Add: Add item to set

```
x = {1,2,3,4}
```

```
x.add(5)
```

```
print x
```

Remove: Remove item from set

```
x = {1,2,3,4}
```

```
x.add(5)
```

```
print x
```

Len: Get length of set

```
x = {1,2,3,4}
```

```
print len(x)
```

Check: Check membership in set

```
x = {1,2,3,4}
```

```
print (1 in x)
```

```
print (4 not in x)
```

Pop: Pop random item from set

```
x = {1,2,3,4}
```

```
print x.pop()
```

```
print x
```

Delete: Delete all items from set

```
x = {1,2,3,4}
```

```
print x.clear()
```

```
print x
```

### Set comprehension:

```
s = {x for x in range(10)}
```

```
print s
```

```
t = {y**2 for y in range(8)}
```

```
print t
```

### Dictionary

- Dictionary is used to represent the group of key, value pairs into a single object.
- Insertion order is not preserved.
- It is a mutable object.
- Duplicate keys are not allowed but values can be duplicate.
- Immutable objects only allowed for keys.
- Generally dictionary represent the “ { } ”

Ex

```
x = {}
```

```
print x
```

```
print type(x)
```

Constructor: Creating new dictionary object

```
x = dict([(1,'apple'),(2,'orange')])
```

```
print x
```

```
z = {'python':100,'java':95,'mysql':90}
```

```
print z
```

Add: Add or change item in dict

```
x = {'apple':25,'dell':34,'lenovo':44}
```

```
x['dell']=24
```

```
print x
```

delete: Remove item in dict at the same time delete the entire object also

```
x = {'apple':25,'dell':34,'lenovo':44}
```

```
del x['dell']
```

```
print x
```

entire object delete

```
x = {'ashok':23,'guido':45,'john':35}
```

```
del(x)
```

```
print x
```

Length: Get length of dict

```
x = {'apple':25,'dell':34,'lenovo':44}
```

```
print len(x)
```

check: Check membership in dict

```
x = {'apple':25,'dell':34,'lenovo':44}
```

```
print ('apple' in x)
```

```
print ('lenovo' not in x)
```

Clear: Delete all items in dict

```
x = {'ashok':23,'guido':45,'john':35}
```

```
print x.clear()
```

Item: This method will return list of items from a dictionary.

Ex

```
x = {'ashok':23,'guido':45,'john':35}
```

```
print x.items()
```

Values: This method will used to return list items in a dictionary.

Ex

```
x = {'ashok':23,'guido':45,'john':35}
```

```
print x.values()
```

Keys: This method is used to return list of keys from given dictionary.

Ex

```
x = {'ashok':23,'guido':45,'john':35}
```

```
print x.keys()
```

### Function

- A function is a block of statements, grouped together with a name.
- We can call this function using function name for any number of times in any module.
- It reduces lines of code.

- **def** keyword is used to create function.

Python supports two types of functions. They are

- Built-in functions
- User defined functions

Built-in functions: The functions which come along with python software are known as pre-defined functions (or) built-in functions.

Ex

- Input ()
- Raw\_input ()
- Type ()
- Id ()
- Len ()

User defined functions: These types of functions are developed by the programmers explicitly to their business requirement are called user defined functions.

Syntax:

```
def function-name(parameters):
```

```
    """    you can write something    """
```

Return value (optional)

Note: within the function syntax, parameters and return statements are optional.

Ex

```
def test():
```

```
print "hello ashok well come to data science"
```

```
test()
```

```
print "hi dear"
```

Parameters: These indicates input of the function i.e if any function contains parameter at the time of calling that function we have to pass the values to those parameters otherwise, we will get error.

Ex

```
def sum(a,b):
```

```
    print "sum of",a, "and",b,"is:",a+b
```

```
sum(10,20)
```

```
sum(2,4)
```

Return Statement: Return statement is used for to come out of method.

- Return statement can be used to return a result back to the calling method.
- If the function doesn't contain return statement then by default it returns none.

Ex-1

```
def add(a,b):
```

```
    return a+b
```

```
z = add(4,6)
```

```
print z
```

Ex-2

```
def who():
```

```
    print "hello ashok how are you"
```

```
print who()
```

### Types of arguments in python

- Required arguments
- Default arguments
- Arbitrary arguments
- Keyword arguments

#### Required Arguments:

Ex

```
def am(a,b):
```

```
    print a+b
```

```
am(2,4)
```

#### Default Arguments:

Ex

```
def john(a="guto martu"):
```

```
    print "the above line mean is good morning in german language",a
```

```
john()
```

```
john("hi well come")
```

#### Arbitrary Arguments:



Ex

```
def arbitrary(*names):  
    #print names  
    for x in names:  
        print x  
arbitrary("ashok","modi","guido")  
arbitrary(10,20,43)
```

### Keyword Arguments:

Ex

```
def keyword(name,msg):  
    print"Hi",name,msg  
keyword(name="harsha",msg="how are you")  
keyword(name="ashok",msg="where are you")  
keyword(name="srinu",msg="what about you")
```

Local Variables: it is a variable whose scope is local (accessibility).

Ex

```
def sum(x,y):  
    sum=x+y  
    return x+y  
print (sum(10,20))
```

Global Variable: A variable whose scope global is known as global variable.

- We can access the outside of the class or thought out the class.

Ex

z=10

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

```
    sum=x+y+z
```

```
    return x+y+z
```

```
print (sum(10,20))
```

Recursive Function calling: Calling one function by the same function is known as recursive function calling.

Ex

```
def fact(num):
```

```
    while num<1:
```

```
        return 1
```

```
    else:
```

```
        f = num * fact(num-1)
```

```
        print f
```

```
        return f
```

```
fact(4)
```

## Lambda

- Lambda function also called as anonymous function because, there is no name for this function.
- It can be defined in a single line.
- It reduces the lack of code.
- It is a special function because we can define this by using “Lambda” keyword.
- It expect min of one argument.

Ex-1

```
mx = lambda x,y:x if x>y else y  
print(mx(8,5))
```

Ex-2

```
x = lambda x:x*2  
print x(4)
```

## Functional Programming

- Functional programming tools are advanced concepts of functions in python.
- Python supports 4-types of functional programming tools
  - Filter
  - Map
  - Reduce
  - Zip

Filter: Filter method is used to get the group of values (required values) from given elements.

- Filter method accepts the data structure as input and it will return same data structure as output.

Ex-2

```
n = [4,3,2,1]
```

```
print(filter(lambda x:x>2,n))
```

Ex-2

```
def even(a):
```

```
    return a%2==0
```

```
print filter(even,range(100))
```

Map: This method is used to perform mathematical operations (manipulation operations) on each & every element of given data structure.

- Map method take a data structure as input and it will return same data structure as output.
- Apply the same operation to the each and every element individually.

Ex-1

```
n = [4,3,2,1]
```

```
print(map(lambda x:x**2,n))
```

Ex-2

```
def data(a):
```

```
    return a*20/5
```

```
print map(data,[1,2,3,4,5,6])
```

Reduce: This method is used to perform mathematical operation on all the elements of given data structure.

- Apply the same operation to items in a sequence.

Ex-1

```
n = [5,6,7,8]
```

```
print(reduce(lambda x,y:x*y,n))
```

Ex-2

```
def add(a,b):
```

```
    return a+b
```

```
print reduce(add,[1,2,3,4])
```

Zip: This method is used to convert the two list into list of tuples.

Ex

```
l1 = ['a','b']
```

```
l2 = [1,2]
```

```
print zip(l1,l2)
```

## [Generators & Iterators](#)

### Generator

- Generators are mainly used to create an object with required values based on requirement.

- We can control the iteration behavior on given elements while working with generators.
- To add any element to an generator we will use “Yield” Keyword.

Ex-1

```
def gen(n):
```

```
    for i in n:
```

```
        yield i
```

```
n=[1,2,3]
```

```
c=gen(n)
```

Ex-2

```
def even(l):
```

```
    for i in l:
```

```
        if i%2==0:
```

```
            yield i
```

```
x = even(range(10))
```

```
print x
```

### Iterator

- In python, iterators are used to iterate a group of elements, contains like list.
- We can create an iterator in python using iter () method.
- Iter () method accepts one parameter i.e, group of elements.
- To access the data from an iterate we will use next method.

Ex

```
x = iter([1,2,3,4,5])
```

```
print x      # checking purpose x.next()
```

## Module

Module:- Every python file itself known as a module. A module can contain classes, variables and functions.

- Every .py file is nothing but one module in python.
- Modules will work like, accessing the called function data from one file into file.

To work with modules we are using two keywords

1. Import
2. from import

Import:- In order to use the properties of one module into another module we use the “ Import ” statement.

Ex-1

```
def add(a,b):    #we save this file maths.py
```

```
    return a+b
```

```
print add(10,20)
```

```
def mul(a,b):
```

```
    return a*b
```

```
print mul(10,20)
```

Ex-2

```
import m
```

```
print maths.add(100,300)
```

**Note:-** Whenever we import one module into another module then imported module compiled file will be generated and stored that file into hard disk permanently.

From..... Import:- In-order to import the specific properties of module into another module we can use “ from.....import “.

Ex

```
from m import add,mul
```

```
print m.mul(10,20)
```

Note:- The above modules are calling custom modules (created by us)

External Modules (Third Party Modules)

- These modules which are available in internet and can be downloaded into our python versions are known as third module.
- We can install any module by using “ pip “.
- Examples paypal, xlrd, django and soapy.

Ex

```
Pip install django == 1.8
```

Built-in Modules :- When we install python automatically available in site-packages.

- Os



- Sys
- Date time
- Re

**OS Module:-** Operating System is software that works as an interface between a user and the computer hardware.

Ex

```
import os
```

```
os.getcwd()          #we are getting the current working directory
```

```
os.chdir('C:\Users\jani\Desktop')  #we are going to one place to
another place
```

```
os.rename('m.py','ma.py')          #changing the name
```

```
os.mkdir('john')                  #we are creating the folder by using mkdir
```

```
os.listdir('C:\\Users\\jani\\Desktop\\ctpwebapp')
```

```
os.path.isdir(isfile)(os.listdir('C:\\Users\\jani\\Desktop\\ctpwebapp'))
```

```
o/p: True
```

**SYS Module:-** sys is a file extension for system file device driver.

Ex

```
sys.path  #we are getting the all paths in current working palce
```

```
sys.path.append("os.listdir('C:\\Users\\jani\\Desktop\\ma.py')") #its
adding the select path into the all paths palce
```

```
sys.version  #we can find the python version throught shell ditectly
```

`sys.byteorder` #it's show the bite order 'little'

`sys.executable` #it find the executable path

`sys.platform` #we find the which platform like windows or linux

## Date Module

### Ex-1

```
import datetime
```

```
print("Current date and time: " , datetime.datetime.now())
```

```
print("Current year: ", datetime.date.today().strftime("%Y"))
```

```
print("Month of year: ", datetime.date.today().strftime("%B"))
```

```
print("Week number of the year: ",  
datetime.date.today().strftime("%W"))
```

```
print("Weekday of the week: ", datetime.date.today().strftime("%w"))
```

```
print("Day of year: ", datetime.date.today().strftime("%j"))
```

```
print("Day of the month : ", datetime.date.today().strftime("%d"))
```

```
print("Day of week: ", datetime.date.today().strftime("%A"))
```

### Ex-2

```
import datetime
```

```
today = datetime.date.today()
```

```
yesterday = today - datetime.timedelta(days = 1)
```

```
tomorrow = today + datetime.timedelta(days = 1)
```

```
print('Yesterday : ',yesterday)
```

```
print('Today : ',today)
```

```
print('Tomorrow : ',tomorrow)
```

Re Module

### Exception Handling

**Exception:** The errors which occur at the time of execution are known as exceptions.

- Generally we get the two types of errors in any programming language, they are
  1. Syntax errors
  2. Runtime errors

#### 1. syntax errors: -

- The errors which occur because of invalid syntax are known as syntax errors.
- Whenever we run program then internally it will check for the syntaxes. If any syntax is written wrongly byte-code will not be generated for the python program.
- Without generating the byte-code program execution will not take place.

Ex

```
def test()
```

```
    print "in m1"
```

#### 2.Runtime errors: -

- The errors which occur at the time of execution of the program are known as runtime errors.
- Generally we will get the run errors because of program logic, invalid input, memory related problems.
- With respect to runtime error corresponding class is available and we call those classes as a exception classes.
  - Key error
  - Index error
  - Zero Division Error
- If the python program does not contain the handle that object then the program will be terminated abnormally.

Abnormal Termination:- The concept of terminating the program In the middle of it's execution without executing last statement of python program is known as abnormal termination.

- Abnormal termination is the undesirable situation in any programming language.

### Exception Handling

- Exception handling is used to stop the abnormal termination whenever error is handling.
- We can implement the exception handling in python by using try and except blocks.

**Try Block:-** Try block should contain the piece of code in which we are expecting exception.

**Note:** if there is no runtime error occurs at the time of execution of the try block then control will not go to the except block.

**Except Block:-** Except block should be proceeded by the try block.

- Except block receives the runtime error representation class object which is given try block and assigns that object to the reference variable of corresponding runtime error representation class.
- In except block we can define the statements to display the user friendly error messages.

Note: Except block will execute only when we got an exception in “try” block.

**Final Block:-** It is optional block used to put up some display string.

Ex-1

```
a = 30
```

```
b = 0
```

```
try:
```

```
    c = a/b
```

```
except Exception as e:
```

```
    print str(e)
```

Note: It show integer division or modulo by zero.

Ex-2

```
try:
```

```
    import ase
```

```
    os.mkdir("abcd1")
```

```
except ImportError:
    print "the given module doesn't exist"
else:
    print "Successfully directory created"
```

Ex-3

```
try :
    a=12
    b=2
    d="abc"
    c=a/b
    c=a/d
    print c
except ZeroDivisionError :
    print "Second number can't be a zero.Enter second number as a non
zero"
except TypeError :
    print " Both values should be numeric but not strings. check the type"
print "Divided successfully"
finally:
    print "do you understand"
```

## User Defined Exceptions:

- User defined exceptions will not raise automatically so that we have to raise those exceptions explicitly.
- We can raise the pre-defined explicitly by using “ raise “ keyword.
- After raising the exception we can handle that exception by using try and except blocks.

Syntax: raise userdefinedexception.

Ex-1

```
raise NameError('ashok wellcome to python world')
```

Ex-2

try:

```
    raise Exception('spam', 'eggs')
```

except Exception as inst:

```
    print "raise error"
```

```
    print type(inst)    # the exception instance
```

```
    print inst.args     # arguments stored in .args
```

```
    print inst          # __str__ allows args to be printed directly
```

```
    x, y = inst.args
```

```
    print 'x =', x
```

```
    print 'y =', y
```

## File Handling

- Programming languages programs memory allocation takes place in RAM area at the time of execution of the program.
- RAM is a volatile so that after execution of the program the memory is going to be de-allocated.
- Programming languages programs are good at processing the data but they cannot store the data in permanent manner.
- After processing the data we have to store the data in permanent manner.

### Python supports following file modes

r --- open a file for reading.

w --- open file for writing creating a new file if it does not exist.

a --- open for appending at the end of the file.

t --- open in text mode.

a --- open in binary mode.

**Read operations:-** When we are working with read operational mode we can use three major functions of text file read operational mode.

- read ()
- readline ()
- readlines ()

read () :- It is used to return complete file content in multi-line string format.

- It does not require any parameter.



Ex

```
fo=open("sample.txt","r")  
print "Name of the file : ",fo.name  
print "Mode of the file : ",fo.mode  
print "Closed or not : ",fo.closed  
fo.close()  
print "after closing, status is : "  
print "Closed or not : ",fo.closed
```

readline () :- It is used to return the first line of the file content.

- It does not require any parameter.

Ex

```
fn=raw_input("Enter a filename to open: ")  
fin=open(fn,"rb")  
str=fin.readline()  
while str :  
    print str  
    str=fin.readline()  
fin.close()
```

readlines ():- It is used to return the first line of the rows.

- To work with read-operational mode the file should exist in local repository.

Ex

```
fn=raw_input("Enter a filename to open: ")
```

```
fin=open(fn,"rb")
```

```
str=fin.readlines()
```

```
print str,type(str)
```

```
fin.close()
```

write Operational Mode:- to work with write operational mode use 'w' as second parameter.

- write ()
- writelines ()

write ():- This method is used to add a single line of string to the file in open method.

- This method requires one parameter.

Ex

```
fp = open("C:\Users\LENOVO\Desktop\ctplinks",'w')
```

```
x = fp.write("hello")
```

```
print x
```

writelines ():- It is used to add multiple lines of data to text file.

- It accepts one parameter.

Ex

```
fp = open("C:\Users\LENOVO\Desktop\ctplinks",'w')  
x = fp.writelines("hello when are you come to bengalore \n hi")  
print x
```

Note:

- Whenever we open the file by default file pointer points the Zero location.
- By using tell() function we can know the current file pointer location.
- By using seek() we can change the file pointer from one location to another location.

### Decorator

- A python decorator is a specific change that we make in python syntax to alter functions easily.
- Decorators and Wrappers in Python. Python has a really nifty language feature that I use and like a lot - decorators (or wrappers).
- They're great for writing a wide variety of reusable code.

Ex

#without using decorator is general way

```
import time
```

```
def calc_square(numbers):
```

```
    start = time.time()
```

```
result = []  
  
for number in numbers:  
    result.append(number*number)  
  
end = time.time()  
  
print("calc_square took" + str((end-start)*1000) + "mil sec")  
  
return result
```

```
def calc_cube(numbers):  
    start = time.time()  
    result = []  
    for number in numbers:  
        result.append(number*number*number)  
    end = time.time()  
    print("calc_cube took" + str((end-start)*1000) + "mil sec")  
    return result  
  
array = range(1,100000)  
  
out_square = calc_square(array)  
  
out_cube = calc_cube(array)
```

**decorators solve both of these issues**

- code duplication.

- clustering main logic of function with additional functionality(i.e timing our example).

Ex

```
import time
```

```
def time_it(func):
```

```
    def wrapper(*args, **kwargs):
```

```
        start = time.time()
```

```
        result = func(*args, **kwargs)
```

```
        end = time.time()
```

```
        print(func.__name__+" took " + str((end-start)*1000) + "mil sec")
```

```
        return result
```

```
    return wrapper
```

```
@time_it
```

```
def calc_square(numbers):
```

```
    result = []
```

```
    for number in numbers:
```

```
        result.append(number*number)
```

```
    return result
```

```
@time_it
```

```
def calc_cube(numbers):
```

```
result = []  
  
for number in numbers:  
    result.append(number*number*number)  
  
return result  
  
array = range(1,100000)  
  
out_square = calc_square(array)  
  
out_cube = calc_cube(array)
```

## Object-Oriented-Programming

OOPS: oops stands for object oriented programming structure. That means grouping multiple objects together & providing some particular structure to achieve security, extendibility & multi-functionality.

- In oops classes and objects are created.
- Code debugging becomes very easy.
- Another main advantage in oops that the programmer can create new classes from existing classes.

### Class

- A class represents common behavior of group of objects.
- A class also contains variables and methods.
- Without class we can't create the object.
- Class doesn't exist physically.

## Object

- Object is used for allocating memory space for non-static variable of a class at run time dynamically is known as object.
- We can create 'n' number of objects for one class.
- object is the blue print of the class
- object exist physically
- without object we can create class

## Encapsulation:

- Taking data and methods as a single unit is called encapsulation.
- A class is an example for the encapsulation.

Ex

class Cust:

```
    cname="ICICI"
```

```
    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,want):
```

```
        self.cbal=self.cbal-want
```

```
def display(self):  
    print self.cname  
    print self.cacno  
    print self.cadd  
    print self.cbal  
    print self.cbname
```

```
c1=Cust("ashok","macherla",1001,100000)
```

```
c1.deposit(10000)
```

```
try:
```

```
    c1.withdraw(100000)
```

```
except:
```

```
    print"your account have insufficient balance"
```

```
c1.display()
```

Static variables (class variables):

- These variables which are declared within the class, outside of all the methods are known as static (or) class variables.
- The data which is common for every object is recommended to represent by using static variables.
- For all the static variables of a class, memory will be allocated only once.



- Static variables of one class we can access within the same class (or) outside of the class by using class name.

Ex

class A:

```
i = 100    # static variable
```

```
def test(self):
```

```
    print obj.i
```

```
obj = A()
```

```
obj.test()
```

```
print obj
```

Non – Static Variables:

- The variables which are declared with self (or) reference variable are known as non-static variables (or) instance variables.
- These variables we can define within the constructor (or) within the method.
- The data which is separate for every object is recommended to represent by using non-static variables.
- For all the non-static variables of a class, memory will be allocated whenever we create object.
- We can access within the same class using 'self'.
- We can access the outside the class by using reference variable name.

Ex

```
class Ashok:
    def ch(self):
        self.i = 100
        self.j = 14.14
    def display(self):
        print self.i
        print self.j
```

```
x = Ashok()
```

```
x.ch()
```

```
x.display()
```

```
x.i = 200
```

```
x.j = 212.2
```

```
x.display()
```

### Constructor:

- Constructor is a special kind of method which executes automatically whenever we create object (without calling it).
- A constructor does not return any value, not even void.
- In python `__init__` method is known as constructor.

Ex-1

```
class Ashok:
    def __init__(self):
```

```

    print "python is the most powerfull programming language"

def test(self):

    print "hello"

obj=Ashok()

obj.test()

```

Ex-2

```

class A:

    def __init__(self,i,j):

        self.i = i

        self.j = j

a = A(14,4)

print a.i

print a.j

```

**All Methods Normal Method, Class Method and Static Method**

Normal Class Method	@Class Method	@Static Method
class A: def x(self): here self should be the first parameter.	class A: @classmethod def x(cls): here cls should be the first parameter, the functionality of self & cls is same.	class A: @staticmethod def x(): here no parameter required.

obj = A() obj.x () here we are creating object.	print A.x() here no need to create object. We can call by class method.	print A.x() same as class method.
--	--	--------------------------------------

Ex-1

#staticmethod

class Myclass:

    @staticmethod

    def the\_\_static\_\_method(x,y):

        print x,y

Myclass.the\_\_static\_\_method(10,20)

Ex-2

#instance(or)non-static method,classmethod,staticmethod

class A:

    def test(self,a,b):#this is the non-static or instance method

        print"this is the instance method"

        return a+b

    @staticmethod

    def the\_\_static\_\_method(a,b):

        print"this is the static method"

```
        return a-b

    @classmethod
    def the__cls__method(cls,a,b):
        print"this is the class method"

        return a*b

obj=A()

print obj.test(10,20)

print A.the__static__method(10,20)

print A.the__cls__method(10,20)
```

Abstraction (or) Data Hiding: The concept of hiding the properties of one class from the other classes or other programs directly is known as data hiding (or) Abstraction.

Ex

```
class Test:
```

```
    def __init__(self,a,b,c):
```

```
        self.a=a
```

```
        self._b=b
```

```
        self.__c=c
```

```
    def test1(self):
```

```
    print self.a
    print self._b
    print self.__c
obj=Test(2,3,4)
print obj.a
print obj._b
print obj._Test__c
#private method
class X:
    __p=1000
    def m1(self):
        print "in m1 of x"
x1=X()
x1.m1()
print x1._X__p
```

## Inheritance

- The concept of creating the new class from already existing class.
- Inheriting the one class properties into another class is called inheritance.

- Creating the sub class from super class that all those properties are passed into the sub class.
- A class which is extended by another class is known as 'super class'
- A class which is extending another class is known as 'sub class'

## Types of Inheritance

- Single level inheritance
- Multilevel inheritance
- Multiple inheritance
- Hierarchical inheritance

### Single level inheritance:

- The concept of inheriting properties from only one class into another class is known as single level inheritance.

Ex

class A:

```
def test(self):  
    print"hello python"  
    return "yes i am coming"
```

class B(A):

```
def fi(self):  
    print"hello wellcome to datascience"
```

obj=B()

```
print obj.test()
```

```
print obj.fi()
```

### Multi level inheritance:

- The concept of inheriting properties from multiple classes into single class with the concept of “one after another” is known as multi-level inheritance.

Ex

```
class A:
```

```
    def test(self):
```

```
        print"hello hyderabd"
```

```
class B(A):
```

```
    def test1(self):
```

```
        print"non sense"
```

```
class C(B):
```

```
    def test2(self):
```

```
        print"who are you"
```

```
obj=C()
```

```
print obj.test()
```

```
print obj.test1()
```

```
print obj.test2()
```



## Multiple inheritance:

- The concept of inheriting properties from multiple classes into single class at a time is known as multiple inheritance.
- MRO (method resolution order) it is searching for the class order.
- Mro following left to right order.

Ex

```
class A:
```

```
    def test(self):
```

```
        print "hello srikanth"
```

```
class B:
```

```
    def add(self):
```

```
        print "hello ashok"
```

```
class C(A,B):
```

```
    def sub(self):
```

```
        print"wellcome to bengulore"
```

```
obj=C()
```

```
obj.test()
```

```
obj.add()
```

```
obj.sub()
```

### Hierarchical inheritance:

- The concept of inheriting properties from one class into multiple class at a time is known as Hierarchical inheritance.

Ex

```
class A:
```

```
    def test(self):
```

```
        print"python is comfortable for datascience"
```

```
class B(A):
```

```
    def test1(self):
```

```
        print"it is the simple language"
```

```
class C(A):
```

```
    def test2(self):
```

```
        print"python is the most powerfull"
```

```
obj=C()
```

```
print obj.test2()
```

```
obj=B()
```

```
print obj.test()
```

```
print obj.test1()
```

```
print obj.test2()
```

## Polymorphism

- Poly means many and morphism means forms.
- Forms mean functionality or logics.
- The concept of defining multiple operations are perform the single object is known as polymorphism.
- Polymorphism can be implemented in python using method overriding.

poly morphism

```
class Animal:
```

```
    def tiger(self):
```

```
        print"geogeious"
```

```
    def cat(self):
```

```
        print"mmow"
```

```
    def dog(self):
```

```
        print"bbow"
```

```
obj=Animal()
```

```
obj.tiger()
```

```
obj.cat()
```

```
obj.dog()
```

Note: python doesn't s support the method overloading, so overcome these problem we are following below methodology.

Ex

Overloading: The concept of defining multiple methods with the same name, different number of parameters with in a same class in known as method overloading.

```
class A:
```

```
    def test(self,x,y):
```

```
        print x+y
```

```
    def test(self,x,y,z):
```

```
        print x+y+z
```

```
obj=A()
```

```
print obj.test(10,20,30)
```

#over loading for all number and string format also accept

```
class A:
```

```
    def f1(self,*args):
```

```
        for p in args:
```

```
            print p
```

```
obj=A()
```

```
obj.f1(10)
```

Overriding: The concept of defining multiple methods with the same name, same number of parameters. one is in super class and another in sub class is known as method overriding.

Ex

```
class A:
```

```
    def f1(self):
```

```
        print "hi"
```

```
class B(A):
```

```
    def f1(self):
```

```
        print"hello"
```

```
obj=B()
```

```
obj.f1()
```

Super Method:

- It is an built-in function in python which is used for calling the super class members, if base class and child class having the same object.

Ex

```
class A(object):
```

```
    def add(self,x,y):
```

```
        return x+y
```

```
class B(A):
```

```
def add(self,x,y):  
    print super(B,self).add(10,20)  
    return x+y  
  
obj=B()  
  
print obj.add(100,200)
```

## Multi-Threading

Thread is functionality or logic which can executed simultaneously along with the other part of the program.

- 'Thread' is a pre-defined class which is defined in 'Threading' Module.
- We can define the logic or functionality as a thread by overriding 'run' method of thread class.
- After defining the functionality or logic, we can execute that logic as a thread by calling 'start()' method of thread class.

Simple Example For Thread:

```
import threading  
  
class Thread1(threading.Thread):  
    def run(self):  
        i=1  
        while i<=10:  
            print i  
            i=i+1
```

```
class Thread2(threading.Thread):
```

```
    def run(self):
```

```
        j=10
```

```
        while j<=20:
```

```
            print j
```

```
            j=j+1
```

```
t1=Thread1()
```

```
t2=Thread2()
```

```
t1.start()
```

```
t1.join()
```

```
t2.start()
```

```
t2.join()
```

### Thread Scheduling:

Among multiple threads which thread has to start the execution first how much time that thread has to start execute, after allocated time is over which thread has to continue the execution next is a dynamic process.

Ex:

```
import time
```

```
from threading import Thread
```

```
def printer():
```

```
for _ in range(3):  
    time.sleep(4.0)  
    print "hello"  
  
thread = Thread(target=printer)  
  
thread.start()  
  
thread.join()  
  
print "goodbye"
```

### Synchronization:

The concept of avoiding multiple threads to access the same functionality at a time is known as synchronization.

- ✓ Synchronization we apply by calling 'acquire()' method 'release()' Method of thread lock.

Ex:

```
import threading  
  
import time  
  
class MyThread (threading.Thread):  
    def __init__(self, thread_id, name, counter):  
        threading.Thread.__init__(self)  
        self.threadID = thread_id  
        self.name = name
```



```
        self.counter = counter

def run(self):

    print "Starting " + self.name

    threadLock.acquire()

    print_time(self.name, self.counter, 3)

    # Free lock to release next thread

    threadLock.release()

def print_time(thread_name, delay, counter):

    while counter:

        time.sleep(delay)

        print "%s: %s" % (thread_name, time.ctime(time.time()))

        counter -= 1


threadLock = threading.Lock()
thread1 = MyThread(1, "Thread-1", 1)
thread2 = MyThread(2, "Thread-2", 2)

thread1.start()
thread2.start()

print 'waiting to finish the thread'

thread1.join()
```

thread2.join()

## Regular Expressions (Or) Regex

- Regular expressions are used to extract required information from the given data by following patterns.
- Regular expressions also used to check whether the given input data is proper format or not.

### Some Fundamentals of the Regex

^ -> start of the line

\$ -> end of the line

\d -> digit

\D -> any single non-digit

\w -> any alphanumeric

\W -> special characters

\b -> word in binary

. -> it matches any single character

+ -> it matches one or more occurrences of preceding character

Ex:

Get a mail id in a text file:

```
import re
```

```
s="hello i am from ashok@wipro.co and send to harsha@cisco.com"
```

```
x=re.findall(r'\S+@\S+',s)
```

```
print x
```

Write a mobile number verification program:

```
import re
```

```
x="hi my name is ashok , my mobile number is 9999999999 and my  
altrnative number is 8888888888"
```

```
y=re.compile(r'(\d{10})')
```

```
k=y.findall(x)
```

```
print k
```

Find particular word in a string:

```
import re
```

```
x="this is ashokashokashok"
```

```
y=re.compile(r'(ashok){1,2}')
```

```
k=y.search(x)
```

```
print k.group()
```

Email-validation program:

Ex:

```
import re
```

```
email=input("enter any email id")
```

```
match=re.search(r"^[\\w\\.\\w+\\{1,3}\\}+@[\\w\\.\\w]",email)
```

if match:

```
    print "valid mail"
```

else:

```
    print "invalid mail"
```

### Pickling and Un-pickling

Pickling:- Pickle module accepts any python object and converts it into a string representation and dumps it into a file by using dump function.

Ex

```
import pickle
```

```
example_dict = {1:"ashok",2:"srinu",3:"harsha"}
```

```
pickle_out = open("dict.pickle","wb")
```

```
pickle.dump(example_dict, pickle_out)
```

```
pickle_out.close()
```

Un-Pickling:- While the process of retrieving original python objects from the stored string representation is called un-pickling.

- Retrieve the data by load function from object.

Ex

```

pickle_in = open("dict.pickle","rb")

example_dict = pickle.load(pickle_in)

print(example_dict)

print(example_dict[2])

```

### Differences Between 2.x and 3.x

2.x	3.x
Print " abcd "	Print ("hello python")
Print is a statement in python 2	Print is a function in python 3
raw_input (' guido ') input (4)	raw_input (deprecated) input (" hi just imagine ")
Python 2 have map function	In python 3 No map (deprecated)
Python 2 have both range and xrange fnctions	Python 3 have only range and xrange (deprecated)
raise IOError, 'msgs' raise IOError (' msgs ')	raise IOError (' msgs ')
Except Exception, e Except Exception as e	Except Exception as e

**Socket Programming:** This socket program used for find the host name and IP address of your system by using python code.

Ex

```
import socket
```

```
a = socket.gethostname()
```

```
b = socket.gethostname(a)
```

```
print a
```

```
print b
```

## Download The MySQL and Connecting to the Python and Django

### MySQL Installation

First we install the MySQL server version 5.6.38

After we install the “mysql connector for python 2.7” type in google open the first link, then it choose the 64-bit (depends on your system configurations) then click it for download. Open the installation file “Run” then it will ask the license agreement click on right button then click on “Next Button” after choose the “mysql server” and click the execute button then it successfully install, after we set the “user-name and password” for the mysql database, click on next button finally finish it

Note: the above description for the MySql install on windows

### Python with mysql connction:

For this requirement we install the python mysql connector, after the successfully installation you can check it on shell then just we can create small database and check it(import mysql.connector) (Type On Google mysql connector for python 2.7) check it

```
#connecting mysql and get the data
```

```
#for my database mysql.connector for connecting username and  
password="root"
```

```
import MySQLdb

db = MySQLdb.connect(host="localhost", # your host, usually localhost
                     user="john",      # your username
                     passwd="megajonhy", # your password
                     db="jonhydb")      # name of the data base

# you must create a Cursor object. It will let
# you execute all the queries you need
cur = db.cursor()

# Use all the SQL you like
cur.execute("SELECT * FROM YOUR_TABLE_NAME")

# print all the first cell of all the rows
for row in cur.fetchall():

    print row[0]

db.close()
```

### Django with mysql:

For this just we install `$ pip install mysqlclient` or `pip install mysqlclient==1.3.6` (Next, we need to install a MySQL database adapter for Python: the mysqlclient package. In the development enviroment:)

If it is successfully install without any error we need to change the django settings

```
DATABASES = {
```

```
'default': {  
    'ENGINE': 'django.db.backends.mysql',  
    'NAME': 'mysql',  
    'USER': 'root',  
    'PASSWORD': 'root',  
    'HOST': 'localhost',  
    'PORT': '3306',  
}  
}
```

Per checking purpose just you can type on cmd prompt python manage.py runserver(if you are not getting any error successfully install django environment also)