PYTHON PROGRAMING

Python Programing

INTRODUCTION TO PYTHON

Python Programing

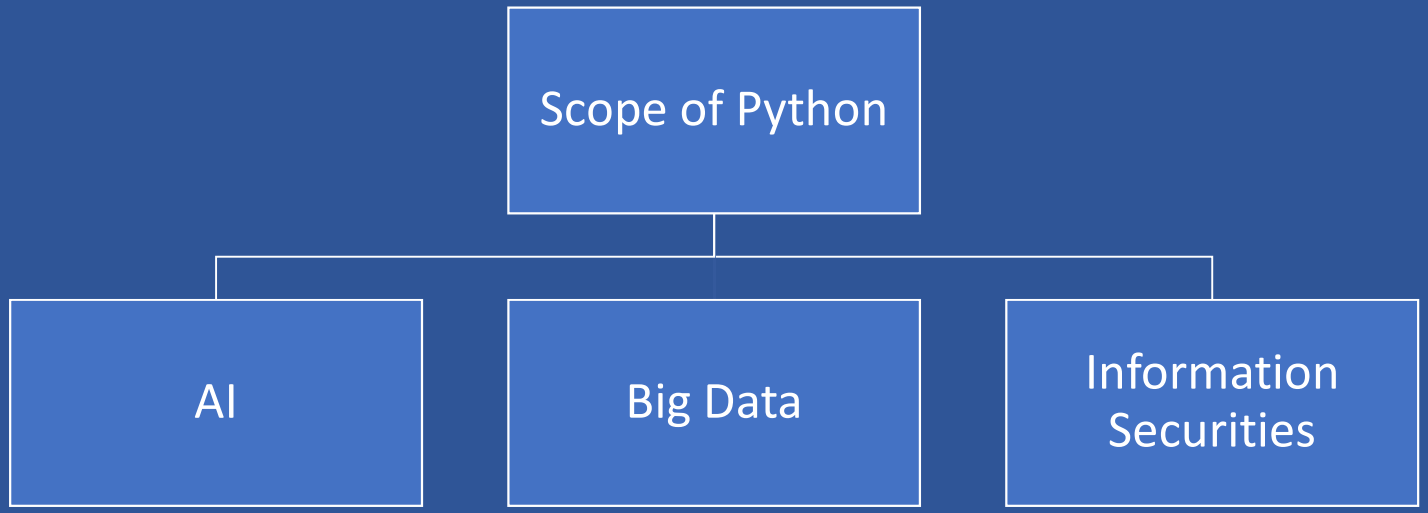
What is Python?

- Python is an interpreted high Level Programing Language
- Python Was Created By Guido van Rossum and its first release was in 1991
- You can develop desktop GUI Applications, Websites and Web Applications using Python which makes it a General Purpose Language.
- Python is Worlds Fastest growing language because of easiness and readability

Python Programing

Why Python?

- Python is more productive than other programing languages
- Companies can optimize their most expensive resource: employees
- Rich set of libraries and frameworks
- Large community



Python Programing

PYTHON SETUP

Python Programing

Software Requirements

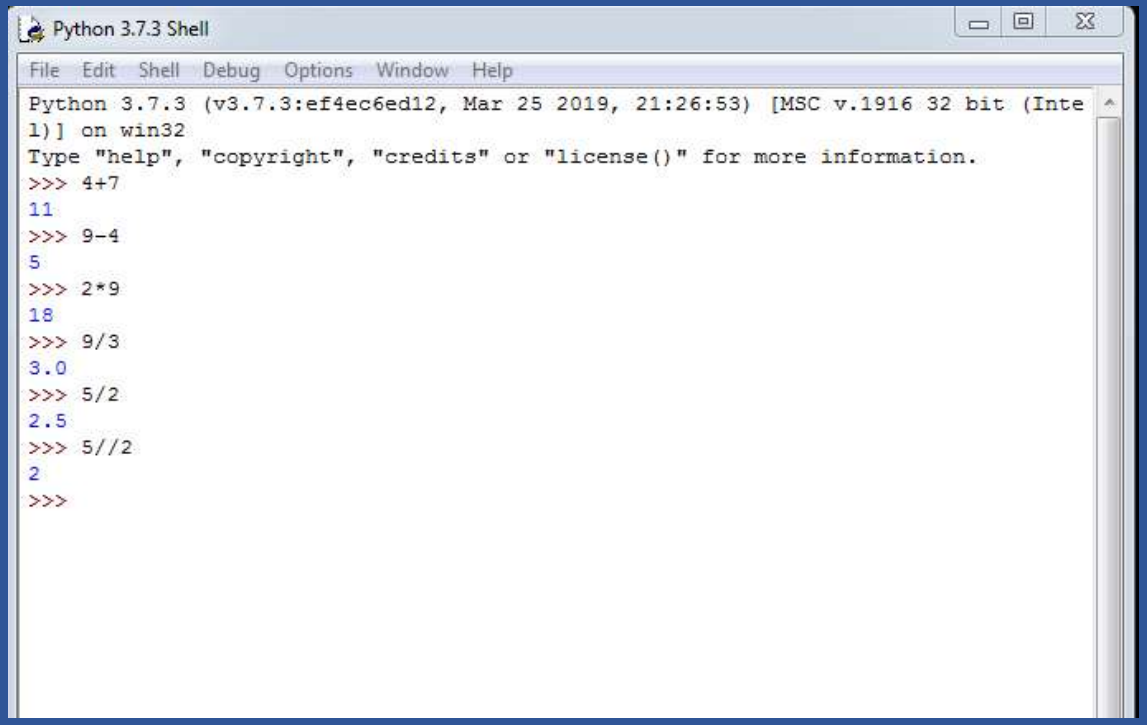
Download Python
https://www.python.org/downloads/

Download Notepad++
https://notepad-plus-plus.org/download/v7.6.6.html

Getting Started With Python

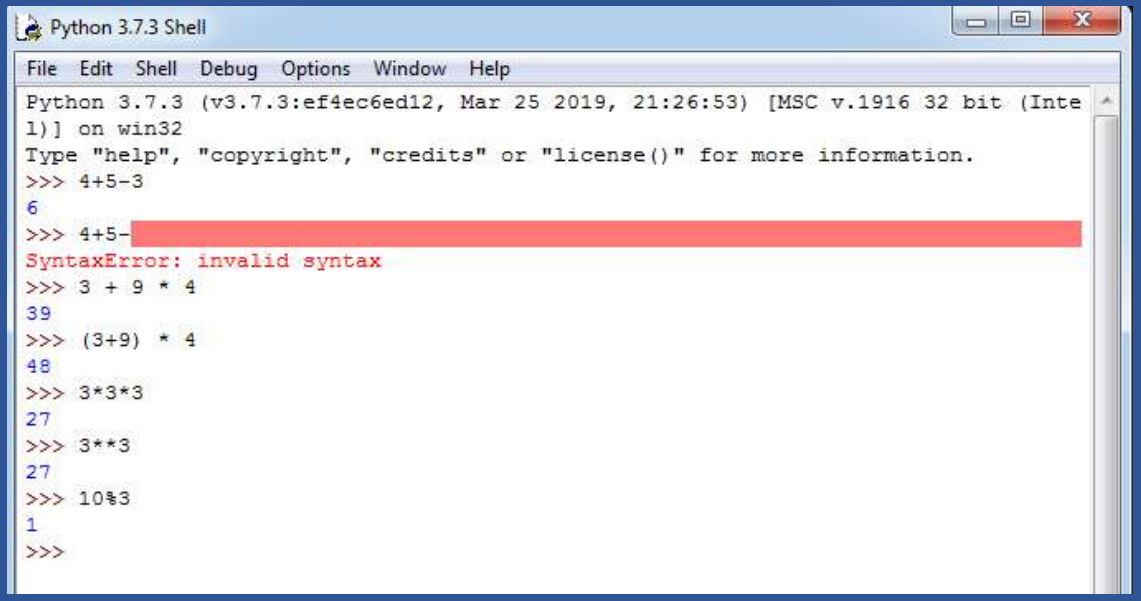
Python Programing

Simple Basics Operations



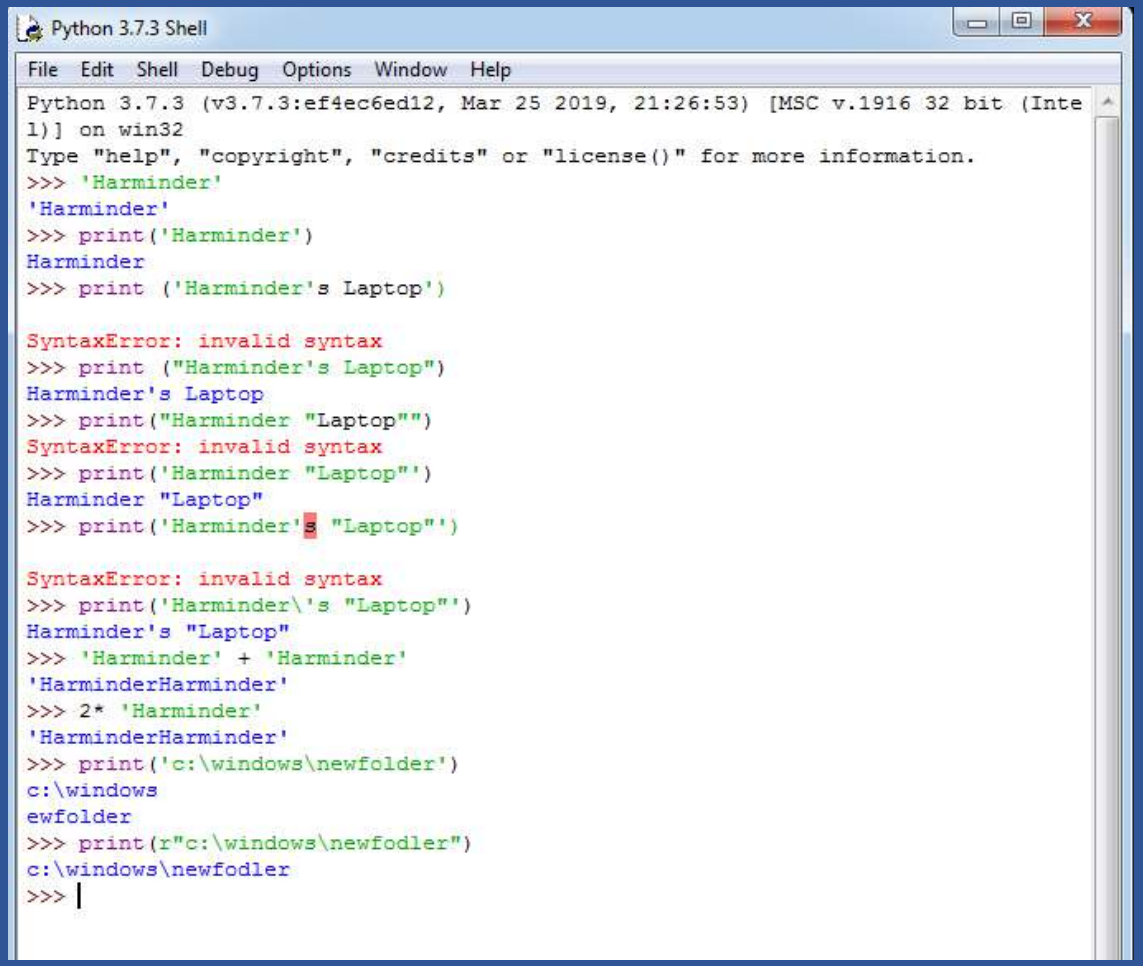
Python Programing

Simple Basics Operations



Python Programing

Simple Basics Operations

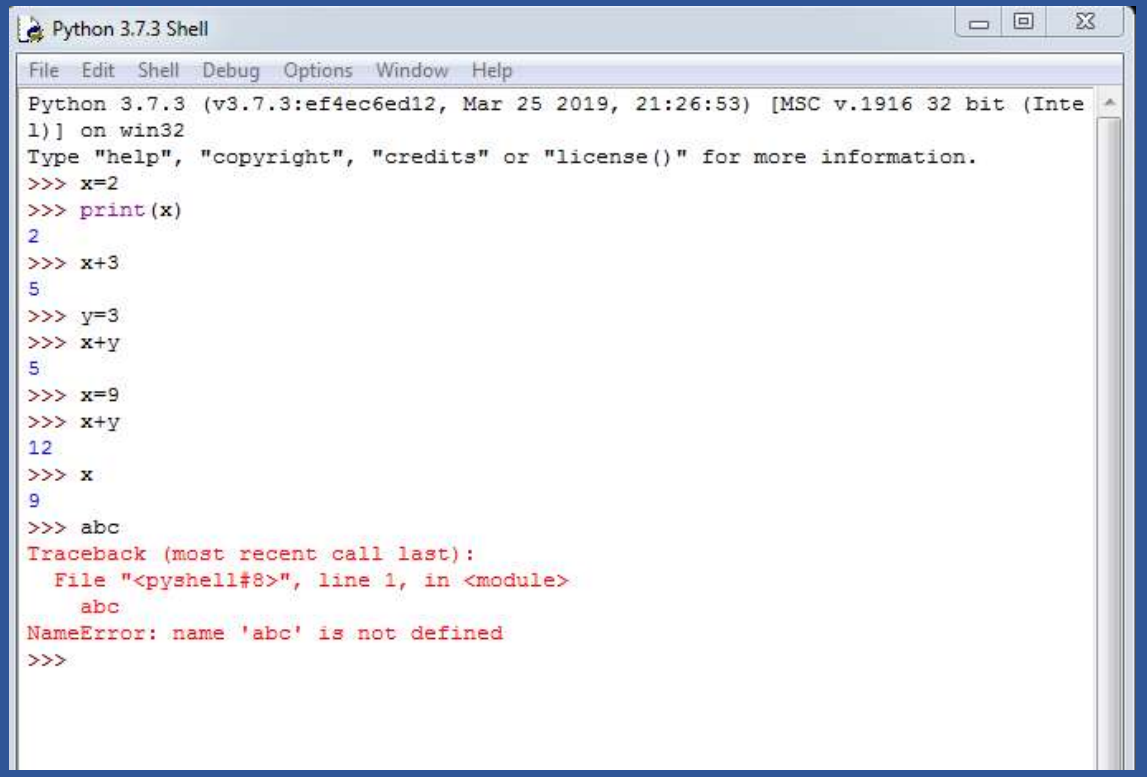


Python Programing

Variables

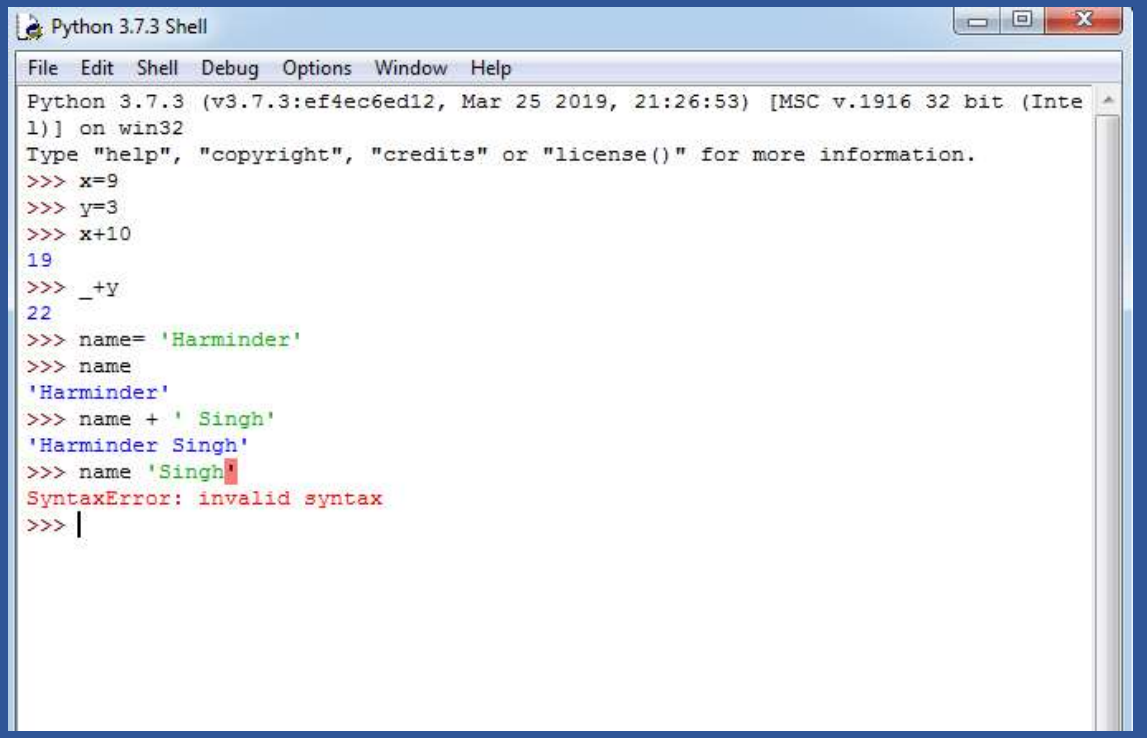
Python Programing

Variables



Python Programing

Variables



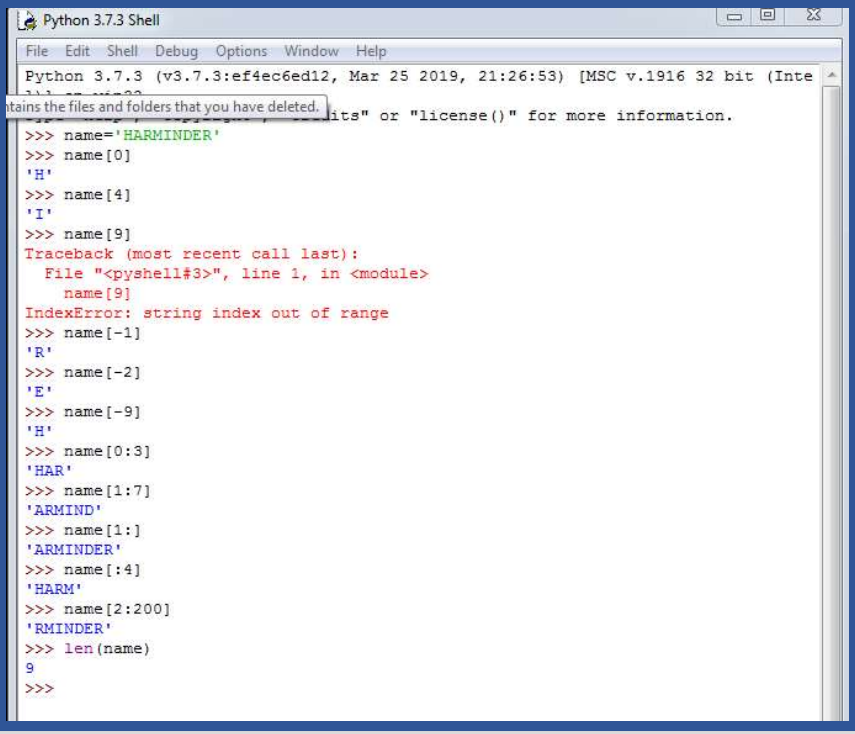
Python Programing

Variables

-9 -8 -7 -6 -5 -4 -3 -2 -1

H A R M I N D E R

0 1 2 3 4 5 6 7 8



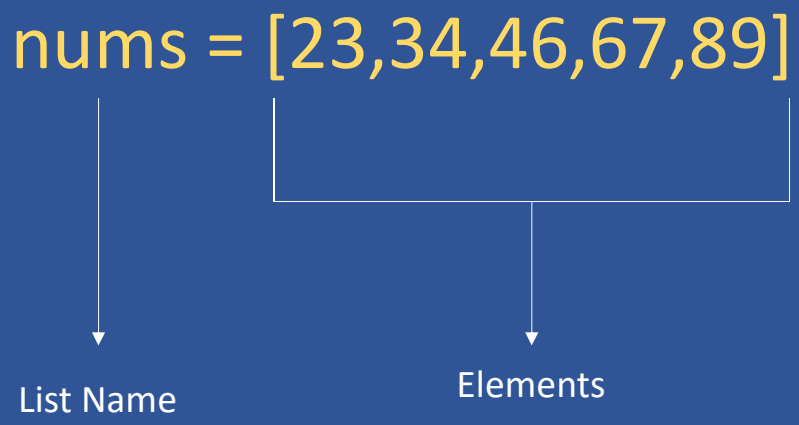
Python Programing

LISTS

Python Programing

LISTS

Defining Lists



Python Programing

LISTS

Accessing Elements

```
          -5  -4  -3  -2  -1
nums = [23,34,46,67,89]
          0   1   2   3   4
```

```
>>nums[1]
34
>>nums[4]
89
>>nums[2:]
[46,67,89]
>>nums[-2]
67
```

Python Programing

LISTS

Accessing Elements

names = [‘Vipul’,’Surender’,’Anup’,’Shubham’]

```
>>names
[‘Vipul’,’Surender’,’Anup’,’Shubham’]
>>names[3]
Shubham
>>names[2:]
[‘Anup’,’Shubham’]
>>names[-2]
Anup
```

LISTS

Lists can have heterogeneous values

```
values = [8.2,'Surender',34]
```

LISTS

Multi Dimensional Lists

```
names = ['Vipul','Surender','Anup','Shubham']
value = [1,2,3,4]
mi = [names,value]
```

```
>>mi
[['Vipul','Surender','Anup','Shubham'] , [1,2,3,4]]
>>mi[0][3]
Shubham
>>mi[1]
[1,2,3,4]
```

LISTS

Lists are Mutable
(Appending a Element)

```
>> nums = [1,2,3,4,5]
>>nums.append(34)
>>nums
[1,2,3,4,5,34]
```

LISTS

Lists are Mutable

(Inserting a Element)

```
>> nums = [1,2,3,4,5]
>>nums.insert(3,45)
>>nums
[1,2,3,45,4,5]
```

LISTS

Lists are Mutable
(Removing a Element)

```
>> nums = [1,2,3,4,5]
>>nums.remove(5)
>>nums
[1,2,3,4]
```

LISTS

Lists are Mutable

(Removing a Element using index)

```
>> nums = [1,2,3,4,5]
>>nums.pop(2)
>>nums
[1,2,4,5]
```

LISTS

Lists are Mutable
(Removing a Element from Last)

```
>> nums = [1,2,3,4,5]
>>nums.pop()
5
>>nums
[1,2,3,4]
```

LISTS

Lists are Mutable
(Removing multiple Elements)

```
>> nums = [1,2,3,4,5]
>>del nums[0:2]
>>nums
[3,4,5]
```

LISTS

Lists are Mutable
(Adding multiple Elements)

```
>> nums = [1,2,3]
>>nums.extend([4,5,6])
>>nums
[1,2,3,4,5,6]
```

LISTS

Lists are Mutable

(Searching Min Value in a List)

```
>> nums = [23,19,85,13]
>>min(nums)
13
```

LISTS

Lists are Mutable

(Searching Max Value in a List)

```
>> nums = [23,19,85,13]
>>max(nums)
85
```

LISTS

Lists are Mutable

(Calculate Sum of a List)

```
>> nums = [23,19,85,13]
>>sum(nums)
140
```

LISTS

Lists are Mutable

(Sorting a List)

```
>> nums = [23,19,85,13]
>>nums.sort()
>>nums
[13,19,23,85]
```

LISTS

Lists are Mutable
(Sorting a List Descending)

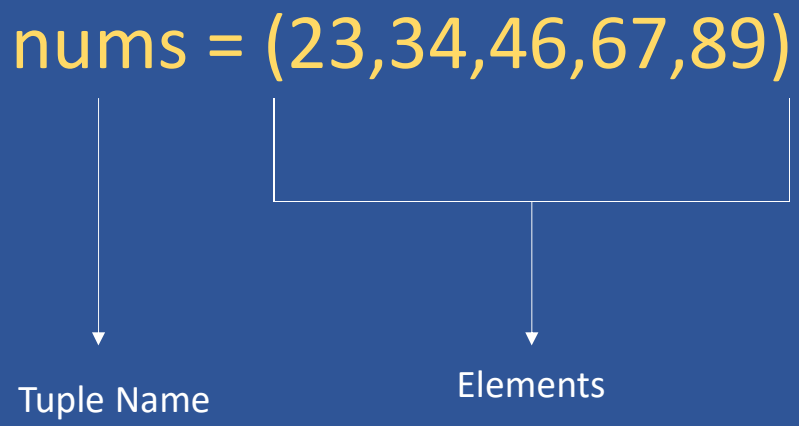
```
>> nums = [23,19,85,13]
>>nums.sort(reverse=true)
>>nums
[85,23,19,13]
```

TUPLE

Python Programing

TUPLE

Defining a Tuple



Python Programing

TUPLE

Accessing Elements

```
          -5  -4  -3  -2  -1
nums = (23,34,46,67,89)
           0   1   2   3   4
```

```
>>nums[1]
34
>>nums[4]
89
>>nums[2:]
[46,67,89]
>>nums[-2]
67
```

TUPLE

Accessing Elements

names = (‘Vipul’,’Surender’,’Anup’,’Shubham’)

```
>>names
(‘Vipul’,’Surender’,’Anup’,’Shubham’)
>>names[3]
Shubham
>>names[2:]
(‘Anup’,’Shubham’)
>>names[-2]
Anup
```

TUPLE

Tuple can have heterogeneous values

values = (8.2,'Surender',34)

TUPLE

Multi Dimensional TUPLE

```
names = ('Vipul','Surender','Anup','Shubham')
value = (1,2,3,4)
mi = (names,value)
```

```
>>mi
(('Vipul','Surender','Anup','Shubham') , (1,2,3,4))
>>mi[0][3]
Shubham
>>mi[1]
(1,2,3,4)
```

TUPLE

Tuples are Immutable

```
>> tup = (1,2,3,4,5)
>>tup[1] = 36
```

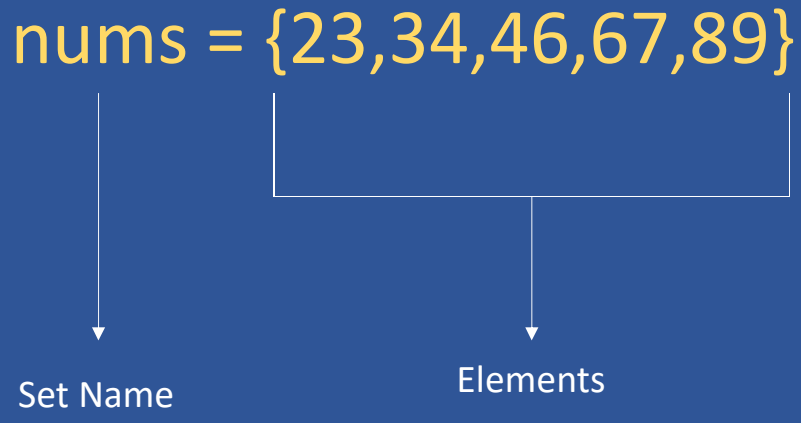
```
>>> tup[1] = 36
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    tup[1] = 36
TypeError: 'tuple' object does not support item assignment
>>>
```

SETS

Python Programing

SETS

Defining a SET



SETS can have heterogeneous values

SETS

values = {8.2,'Surender',34}

Python Programing

SETS

Check if element exists in SET

```
values = {8.2,'Surender',34}
print("surender" in values)
```

SETS

Adding Element to SETS

```
values = {8.2,'Surender',34}
Values.add("hello")
```

SETS

Adding Multiple Element to SETS

```
values = {8.2,'Surender',34}
values.update([3,4,5])
```

SETS

Removing Element From SETS

(Gives an Error when Item is not in Set)

```
values = {8.2,'Surender',34}
values.remove('Surender')
```

SETS

Removing Element From SETS

(No Error when Item is not in Set)

```
values = {8.2,'Surender',34}
values.discard('Surender')
```

Removing Random Element From SETS

SETS

```
values = {8.2,'Surender',34}
values.pop()
```

SETS

Clearing SET

```
values = {8.2,'Surender',34}
values.clear()
```

SETTING PATH FOR WINDOWS

Python Programing

Checking If already Set

Setting Path for Windows

Python Programing

Setting Path for Windows

Copy Following Paths

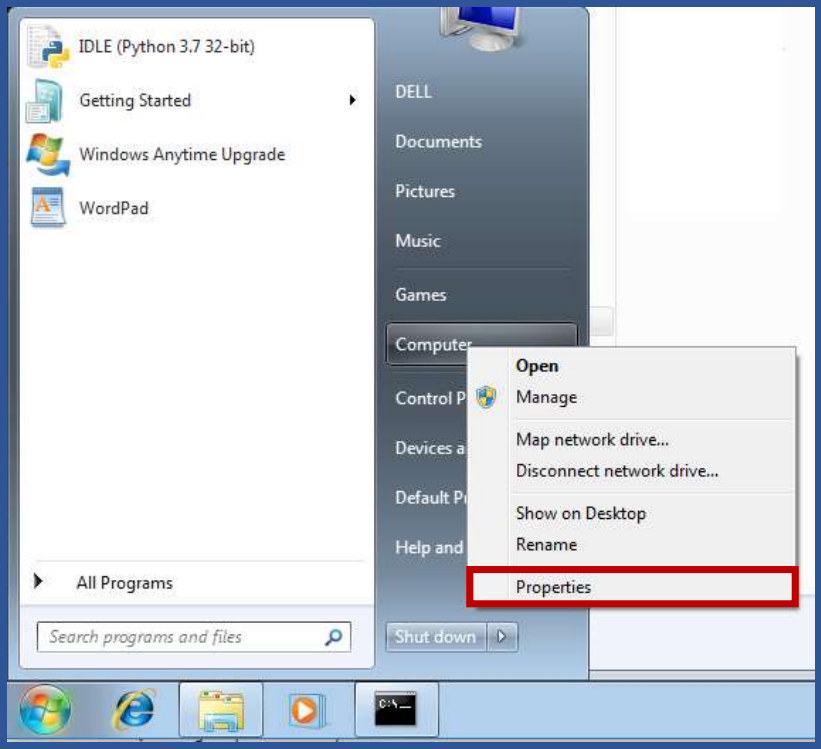
C:\Users\Your_Username\AppData\Local\Programs\Python\Python37-32

C:\Users\ Your_Username \AppData\Local\Programs\Python\Python37-32\Scripts

Python Programing

Setting Path for Windows

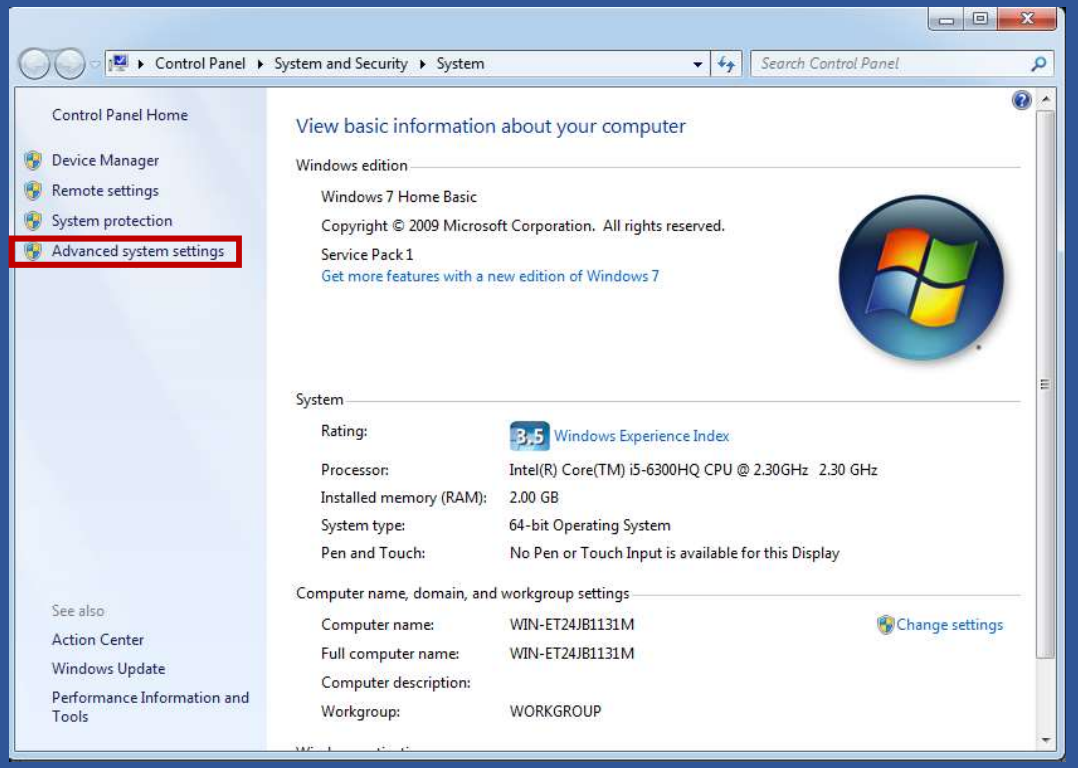
Go to Following Path



Python Programing

Setting Path for Windows

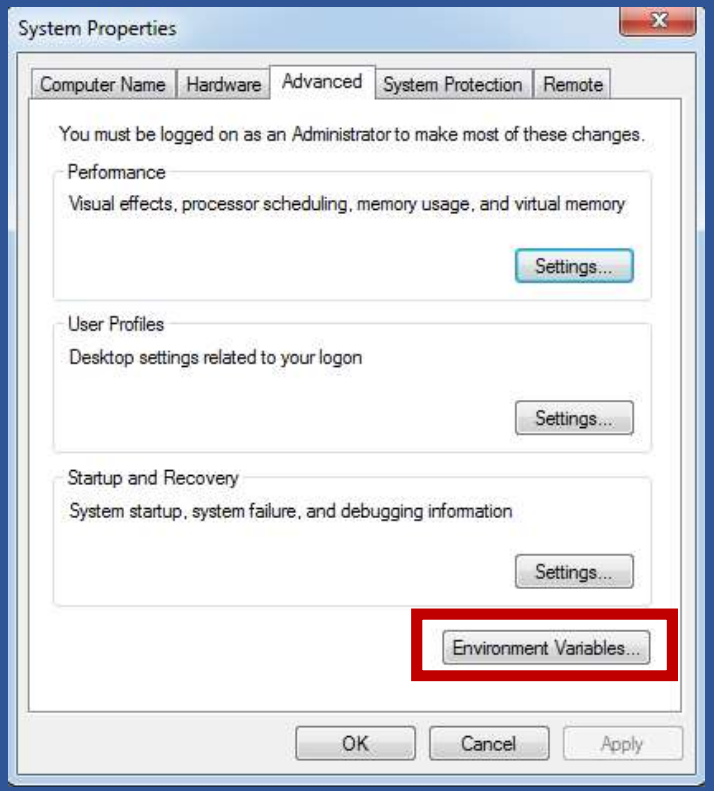
Go to Following Path



Python Programing

Setting Path for Windows

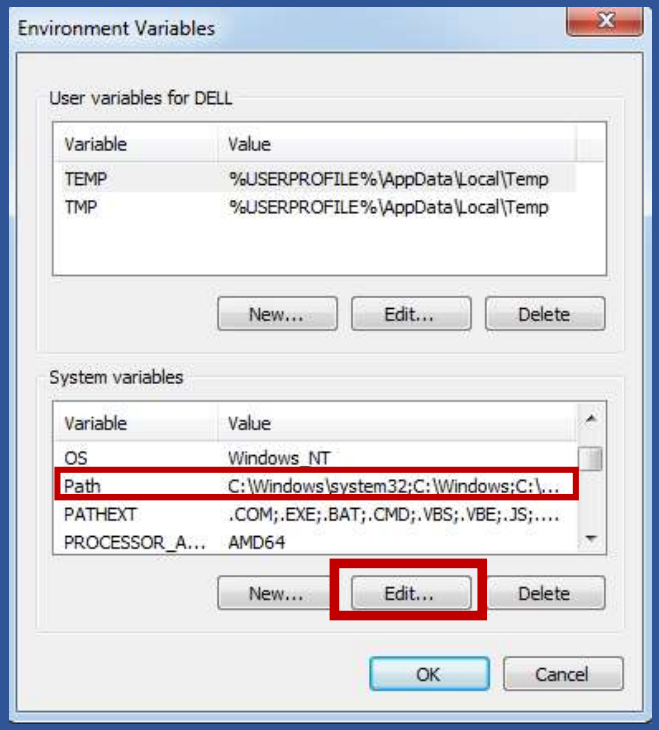
Go to Following Path



Python Programing

Setting Path for Windows

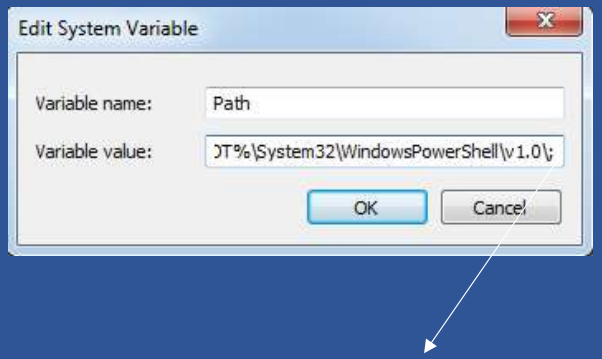
Go to Following Path



Python Programing

Setting Path for Windows

Go to Following Path

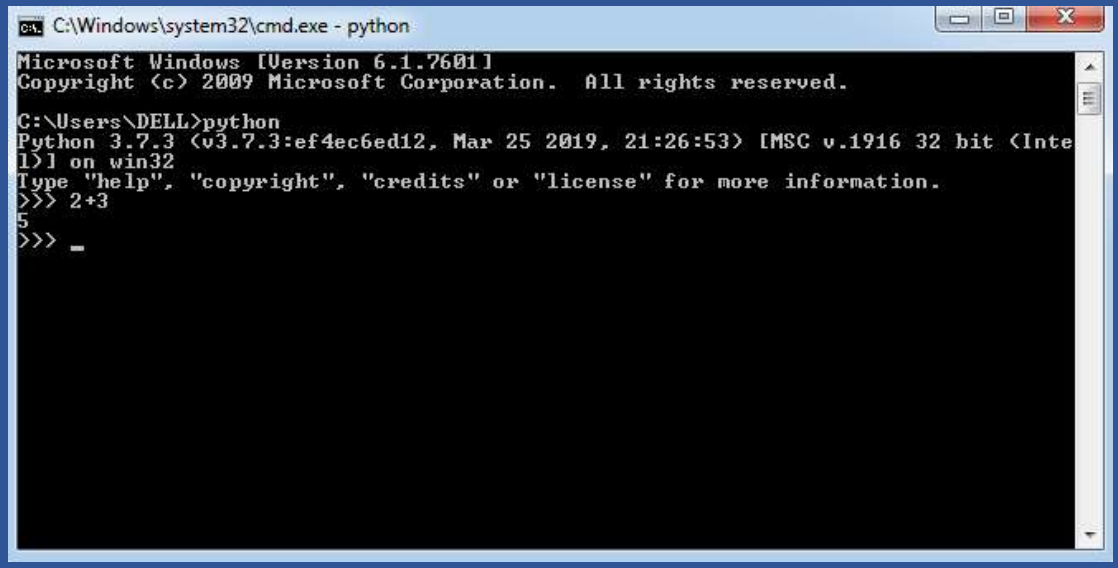


1. Copy both the paths after this semicolon
2. Separate both paths with semicolon

Python Programing

Verification

Setting Path for Windows



Python Programing

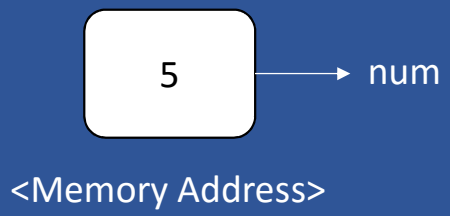
Variable Memory Concept

Python Programing

Variable
(Memory Concept)

Variable Storage

num=5



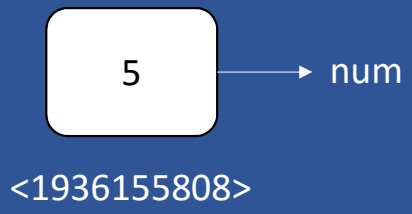
Python Programing

Variable

(Memory Concept)

Getting Address

```
>>num=5
>>Id(num)
1936155808
```

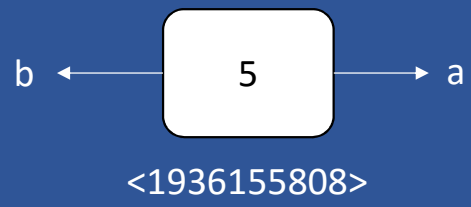


Variable

(Memory Concept)

Variables with Same value has same memory Address

```
>>a=5
>>b=5
>>id(a)
1936155808
>>id(b)
1936155808
```



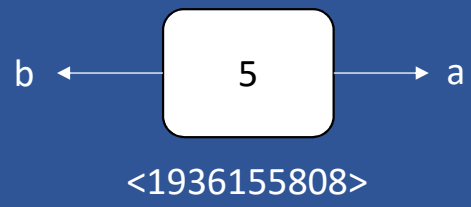
Python Programing

Variable

(Memory Concept)

Variables with Same value has same memory Address

```
>>a=5
>>b=5
>>id(a)
1936155808
>>id(b)
1936155808
```



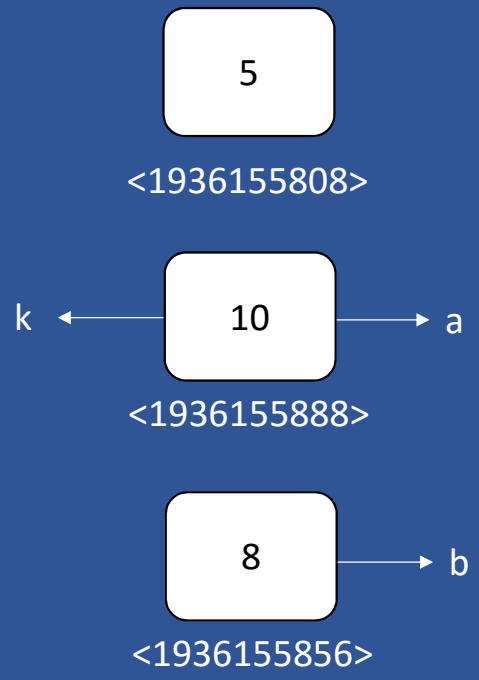
Python Programing

Variable

(Memory Concept)

Concept of Garbage Value

```
>>a=5
>>b=5
>>k=a
>>a=10
>>b=8
>>k=10
```



Python Programing

Variable

(Memory Concept)

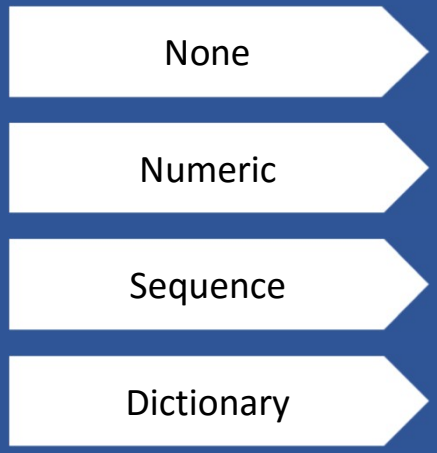
Type of a Variable

```
>>a=5
>>type(a)
<class 'int'>
>>b=4.6
<class
'float'>
```

Data Types

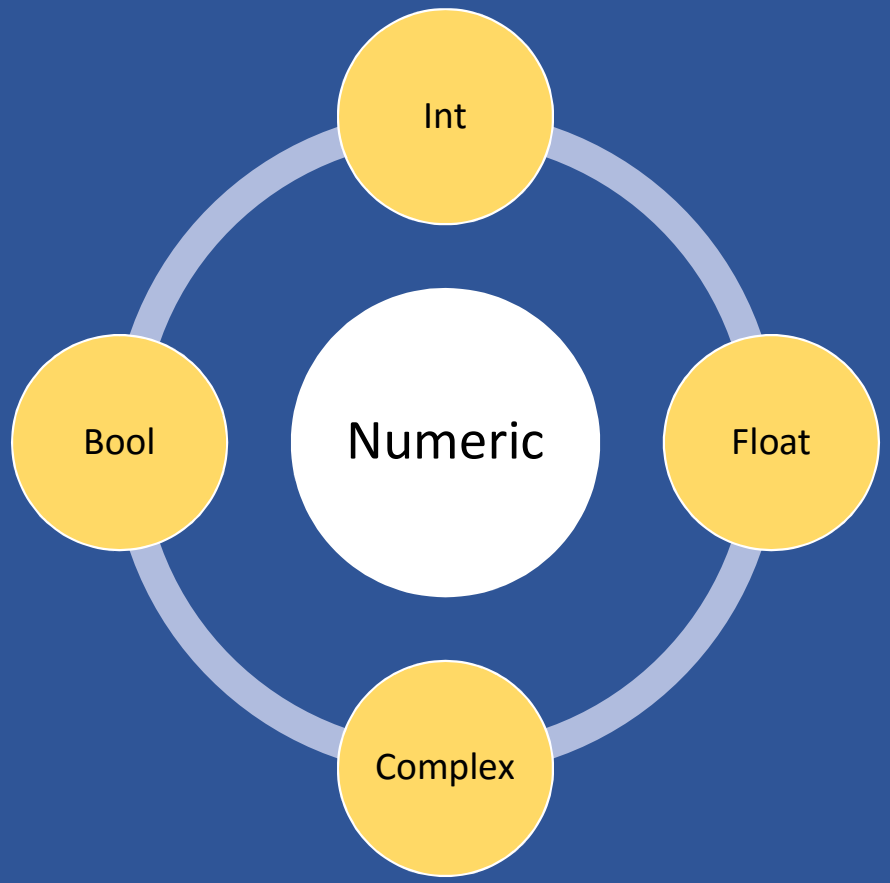
Python Programing

Data Types



Python Programing

Data Types



Python Programing

Data Types

Numeric Examples

INT

```
>>num=5
>>type(num)
<class 'Int'>
```

FLOAT

```
>>num=5.7
>>type(num)
<class 'float'>
```

Complex

```
>>num = 6+9j
>>type(num)
<class 'complex'>
```

BOOL

```
>>a=5
>>b=6
>>a<b
True
```

Data Types

Data Types Conversions

INT → FLOAT

```
>>num=5
>>float(num)
>>num
5.0
```

FLOAT → INT

```
>>num=5.7
>>int(num)
>>num
5
```

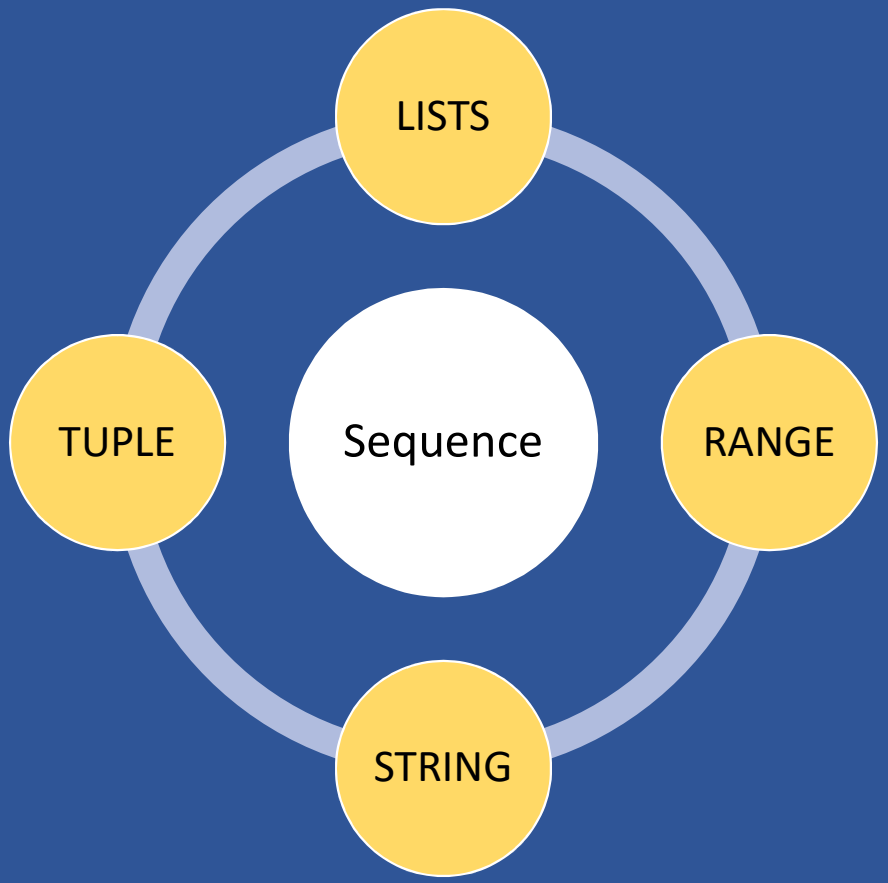
INT → COMPLEX

```
>>a = 6
>>b = 7
>>c = complex(a,b)
>>c
6+7j
```

BOOL → INT

```
>>a=5
>>b=6
>>c = a<b
>>int(c)
1
```

Data Types



Python Programing

Data Types

Sequence Examples

LISTS

```
>>a= [1,2,3,4]
>>type(a)
<class 'List'>
```

TUPLE

```
>>a=(1,2,3,4)
>>type(a)
<class 'Tuple'>
```

STRING

```
>>str = 'Harminder'
>>type(str)
<class 'String'>
```

RANGE

```
>>a=range(0,10,2)
>>type(a)
<class 'Range'>
```

Data Types

Dictionary

Definition

```
>>a= {'name':'Harminder','class':'1st'}
>>type(a)
<class 'Dict'>
```

Accessing Keys

```
>>a= {'name':'Harminder','class':'1st'}
>>a.keys()
dict_keys{['name','class']}
```

Accessing Values

```
>>a= {'name':'Harminder','class':'1st'}
>>a.values()
dict_values{['Harminder','1st']}
```

Accessing Specific Index

```
>>a= {'name':'Harminder','class':'1st'}
>>a['class']
'1st'
```

OPERATORS

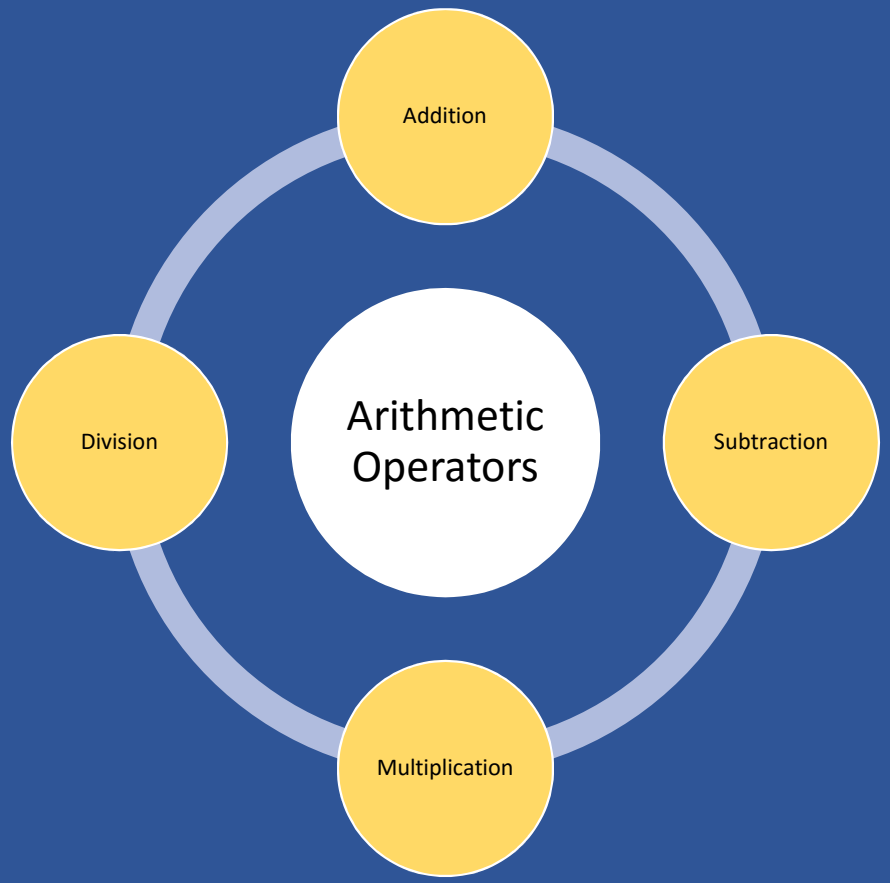
Python Programing

Operators

- Arithmetic Operators
- Assignment Operators
- Relational Operators
- Logical Operators

Python Programing

Operators



Python Programing

Operators

Arithmetic Operators

Addition

```
>>a=5
>>b=6
>>a+b
11
```

Subtraction

```
>>a=5
>>b=6
>>b-a
1
```

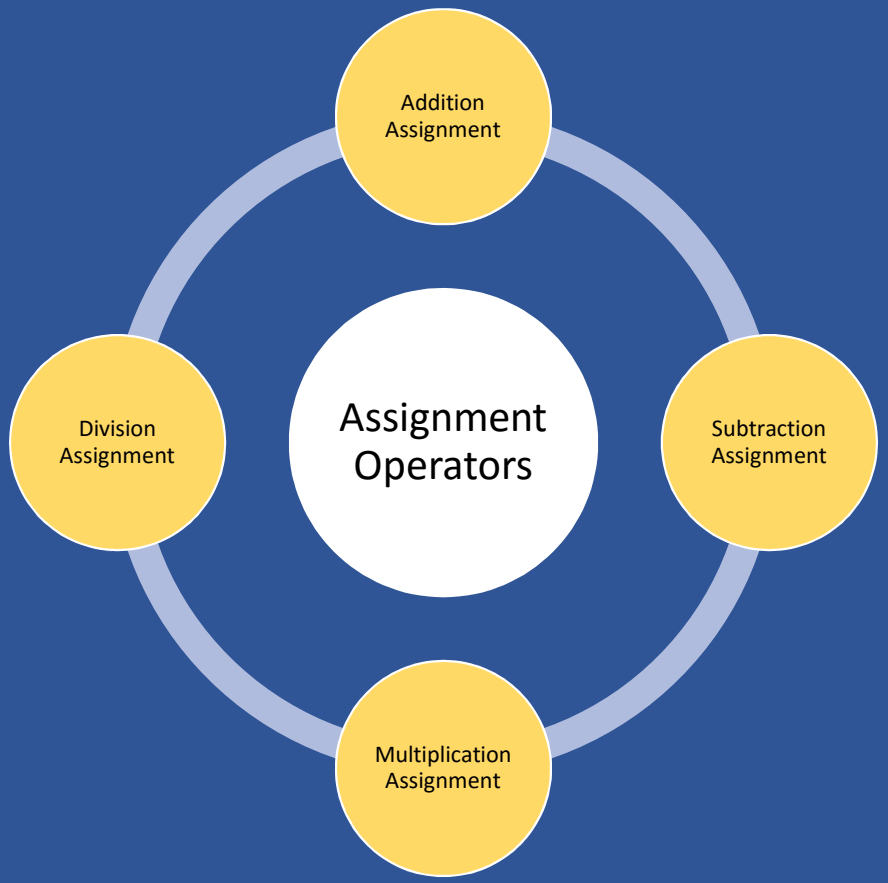
Multiplication

```
>>a=5
>>b=6
>>a*b
30
```

Division

```
>>a=30
>>b=5
>>a/b
6
```

Operators



Python Programing

Operators

Assignment Operators

Addition Assignment

```
>>a=5
>>a += 2
>>a
7
```

Subtraction Assignment

```
>>a=5
>>a-=2
>>a
3
```

Multiplication Assignment

```
>>a=5
>>a*=3
>>a
15
```

Division Assignment

```
>>a=25
>>a /= 5
>>a
5.0
```

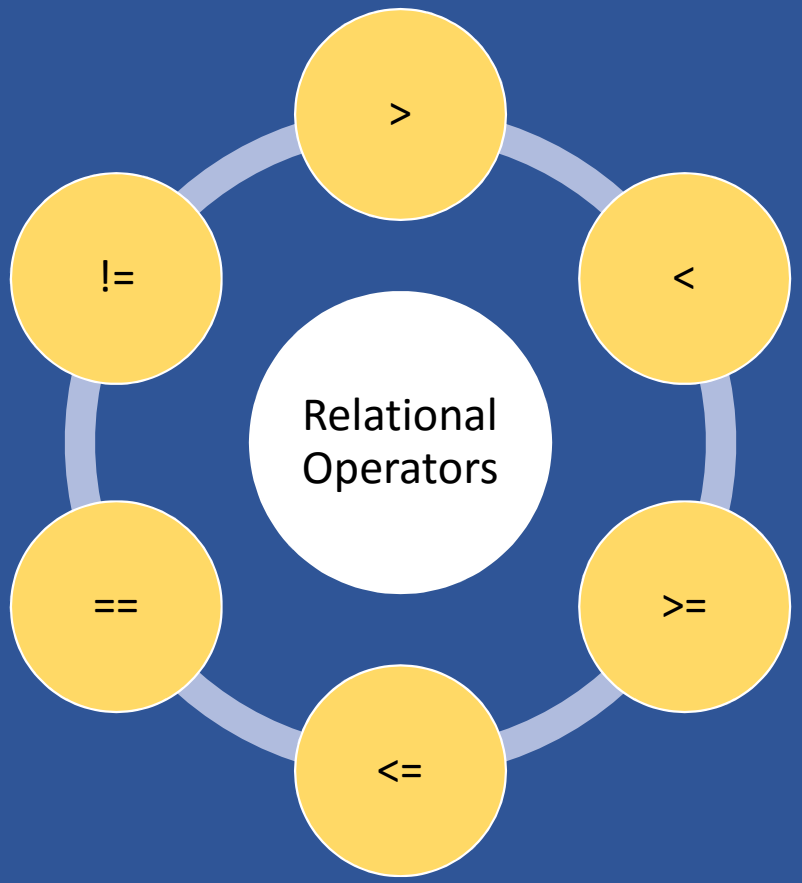
Operators

Assignment Operators

Assigning Multiple Variables at once

```
>>a,b=5,8
>>a
5
>>b
8
```

Operators



Python Programing

Operators

Relational Operators

<

```
>>a=5
>>b=2
>>a<b
False
```

>

```
>>a=5
>>b=2
>>a>b
True
```

>=

```
>>a=5
>>b=2
>>a>=b
True
```

<=

```
>>a=5
>>b=2
>>a<=b
False
```

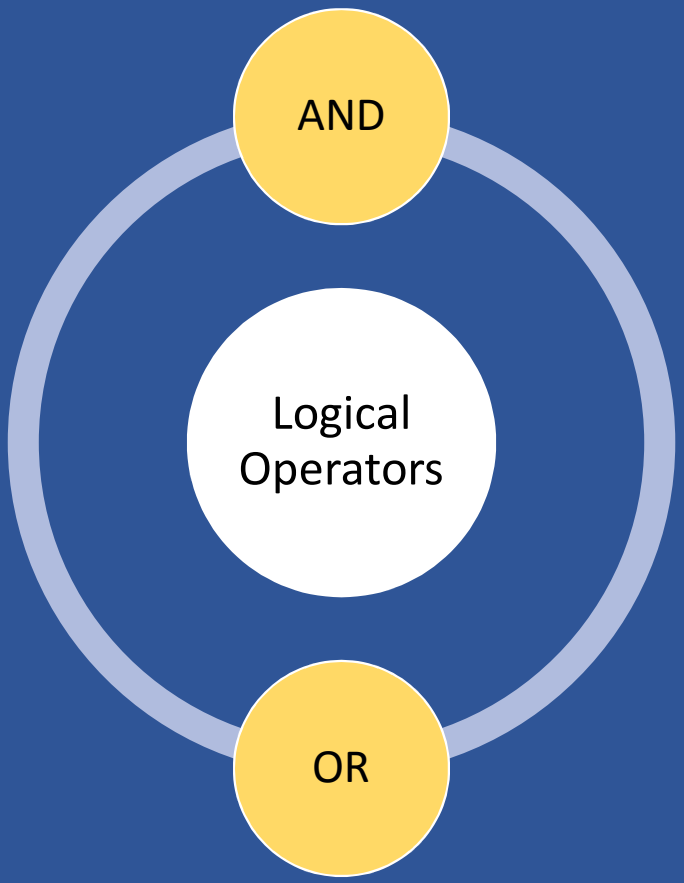
Operators

Relational Operators

==
>>a=5 >>b=5 >>a==b True

!=
>>a=5 >>b=2 >>a!=b True

Operators



Python Programing

Operators

Logical Operators

AND

```
>>a=5
>>b=2
>>a>5 and b=2
False
```

OR

```
>>a=5
>>b=2
>>a>5 or b=2
True
```

BITWISE OPERATOR

Python Programing

Bitwise Operators

- AND (&)
- OR (|)
- XOR (^)
- Left Shift (<<)
- Right Shift (>>)

Python Programing

Bitwise Operators

Decimal to Binary Conversion

12 → 1100

2	12	
2	6	0
2	3	0
	1	1

Python Programing

Bitwise Operators

Binary to Decimal Conversion

1100 → 12

1 1 0 0

$2^3 + 2^2 + 2^1 + 2^0$

8+4=12

Python Programing

Bitwise Operators

Bitwise (AND)

12 & 13 = 12

00001100 -> 12
00001101 -> 13

00001100 -> 12

Bitwise Operators

Bitwise (OR)

12 | 13 = 13

00001100 -> 12
00001101 -> 13

00001101 -> 13

Bitwise Operators

Bitwise (XOR)

12 ^ 13 = 1

00001100 -> 12
00001101 -> 13

00000001 -> 1

Bitwise Operators

Left Shift (<<)

10 << 2 = 40

00001010.000 -> 10
00001010.00.0 -> 40

Bitwise Operators

Right Shift (>>)

10 >> 2 = 2

00001010.000 -> 10
00001010000 -> 2

Math Module

Python Programing

Importing Math Module

Math Module

>>Import math

Python Programing

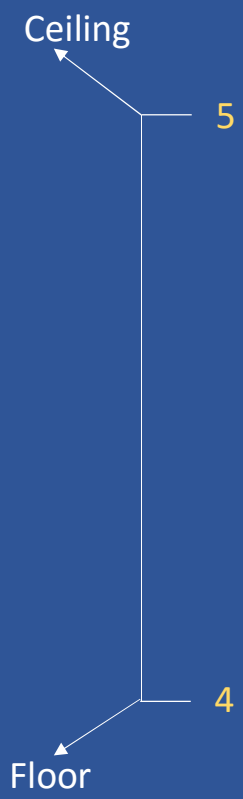
Math Module

Finding Square Root

```
>>Import math
>>x=math.sqrt(25)
>>x
5
```

Math Module

Math Functions



Floor

```
>>x=math.floor(4.9)
>>x
4
```

Ceil

```
>>x=math.ceil(4.1)
>>x
5
```

Power

```
>>x=math.pow(4,2)
>>x
16
```

Python Programing

Alice Math Module

Math Module

```
>>Import math as m
>>x=m.sqrt(25)
>>x
5
```

Importing Specific functions of Math Module

Math Module

>>from math import sqrt

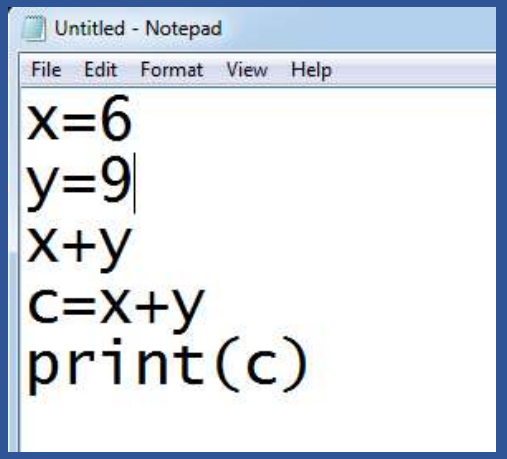
Python Programing

Creating & Running Python Files

Python Programing

Write a Program on Notepad/IDE

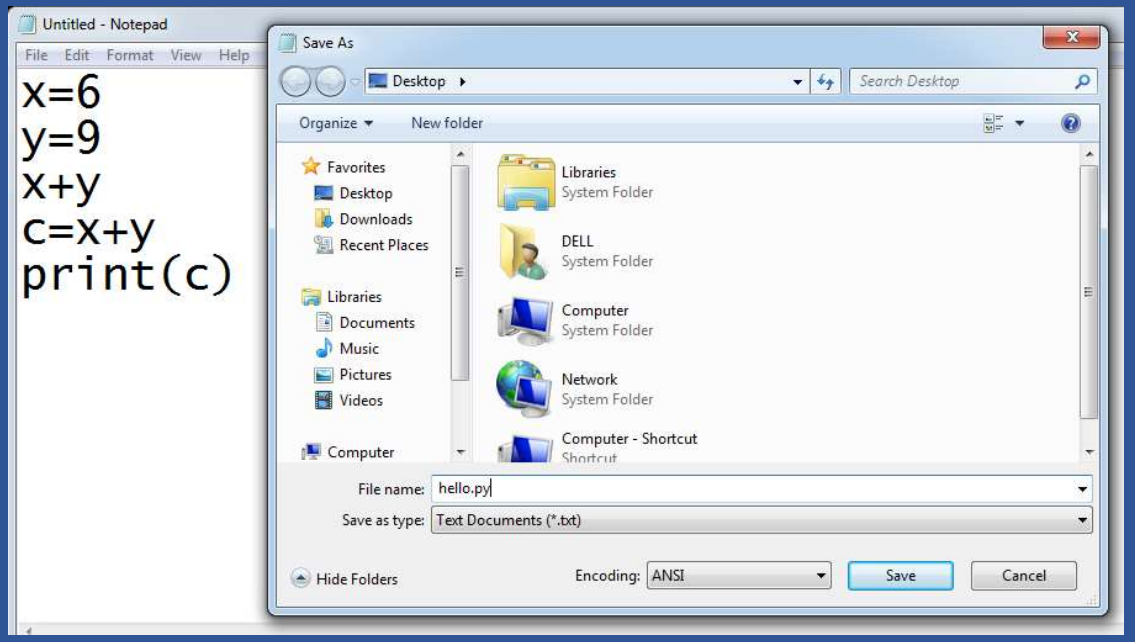
Creating & Running Py Files



Python Programing

Creating & Running Py Files

Save file with PY Extension



Python Programing

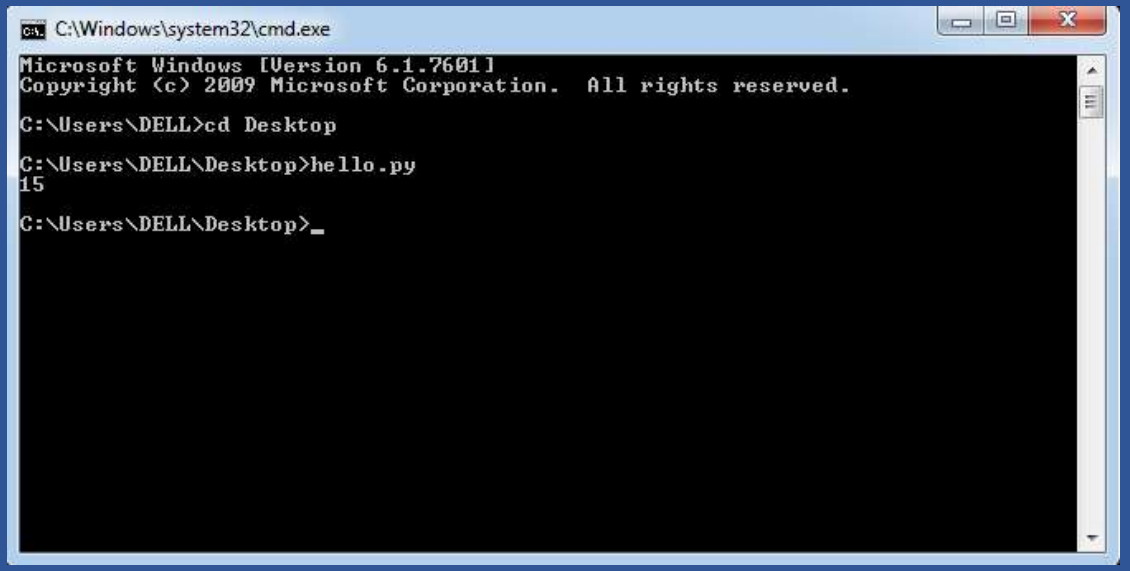
Creating & Running Py Files

Open CMD and Change path to file's location

Python Programing

Creating & Running Py Files

Call the python file



Python Programing

User Input

Python Programing

Input Function

User Input

x=input(“Please Enter Your Input”)

Python Programing

User Input

Input Function only accept strings

```
x=input("Please Enter Your Input")
Print(type(a))
```

```
Please Enter Your Input 1
<class 'str'>
```

User Input

Input Function only accept strings

```
x=input("Please Enter First Number")
y=input("Please Enter Second Number")
c=x+y
print(c)
```

```
Please Enter First Number 1
Please Enter Second Number 2
12
```

User Input

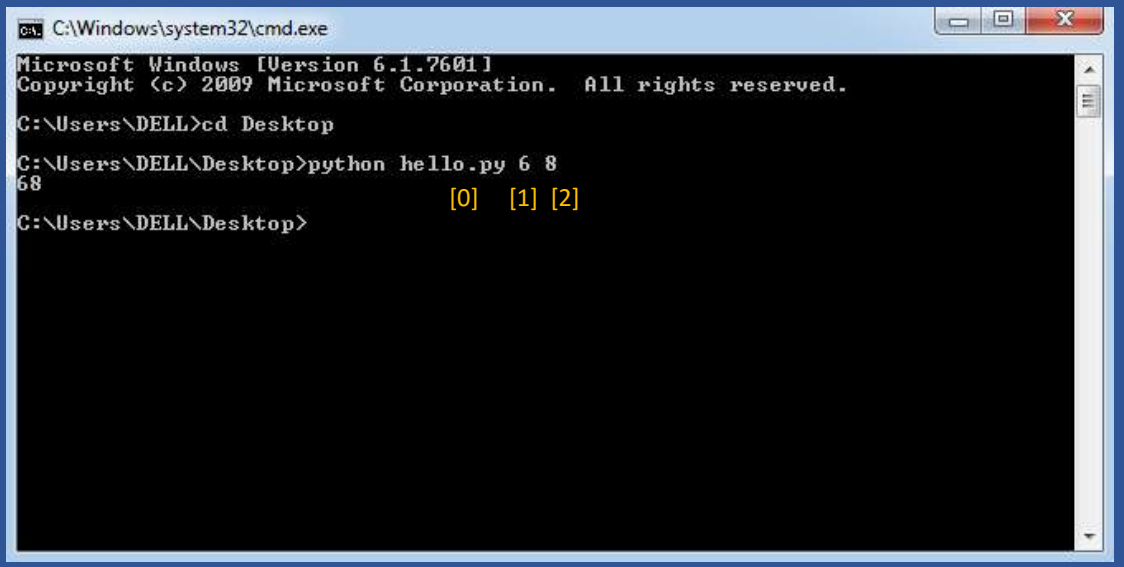
Passing Argument Input in CMD

```
Import sys
x=sys.argv[1]
y=sys.argv[2]
c=x+y
print(c)
```

Python Programing

User Input

Passing Argument Input in CMD

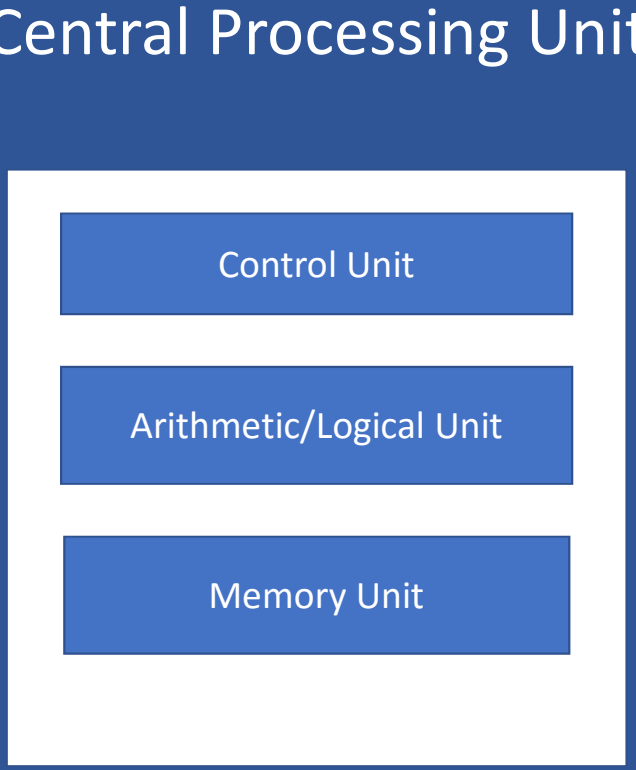


Python Programing

Control Flow Statements

Python Programing

Control Flow Statements



Python Programing

Control Flow Statements

IF Statement

Python Programing

Control Flow Statements

IF Statement Needs Indentation

```
X=5
If x==3:
        print("equal to five")
print("hello")
```

Python Programing

Control Flow Statements

Else Statement

```
X=5
If x==5:
        print("equal to five")
else:
        print("Not Equal")
```

Python Programing

Control Flow Statements

Nested IF Statement

```
X=5
If x>=5:
        print("x is greater")
        if x==5:
                print("x is equal")
else:
        print("x is smaller")
```

Python Programing

Control Flow Statements

Nested IF Statement

```
X=5
If x==1:
        print("One")
elif x==2:
        print("Two")
elif x==3:
        print("Three")
else:
        print("Wrong Input")
```

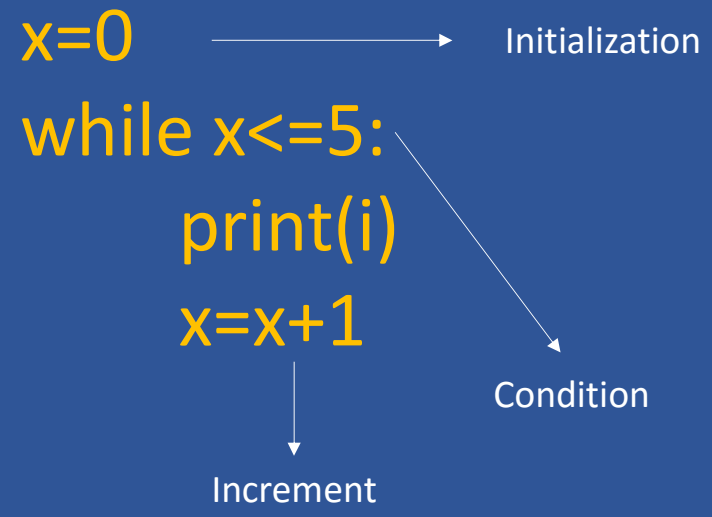
Python Programing

Loops

Python Programing

Loops

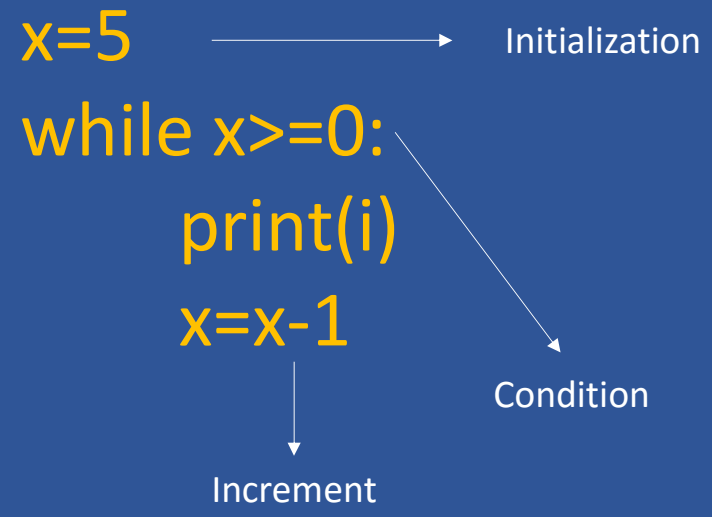
While Loop



Python Programing

Loops

While Loop (Reverse)



Loops

While Loop (Nested)

```
x=0
while x<=5:
        print("Python",end="")
        j=0
        while j<=5:
                print("Rocks",end="")
                j=j+1
        x=x+1
        print()
```

Loops

For Loop with List

```
a = [“Harminder”,1,”Surender”]

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

Loops

For Loop with String

```
a = “Harminder”

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

Loops

For Loop with Tuple

```
a = (“hi”,”harminder”,”surender”)
for i in a:
        print(i)
```

Loops

For Loop with Sets

```
a = {“hi”,”harminder”,”surender”}
for i in a:
        print(i)
```

Loops

For Loop with Range

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

Python Programing

Loops

For Loop with Range

```
for i in range(10,21,1):
print(i)
```

Python Programing

Loops

For Loop with Range

```
for i in range(20,0,-1):
print(i)
```

Python Programing

Loops

Nested For Loop

```
for i in range(5):
        for j in range(5):
                print(j,end="")
        print()
```

Loops

Break Statement

```
for i in range(1,10,1):
        if i==5:
                break
        print(i)
```

Loops

Continue Statement

```
for i in range(1,10,1):
        if i==5:
                continue
        print(i)
```

Loops

Pass Statement

```
for i in range(1,100,1):
        if i%2!=0:
                pass
        else:
                print(i)
```

Loops

For Else

```
a=[1,2,3,4,6,7]
for i in a:
        if i%5=0:
                break
else:
        print("Not Found")
```

Functions

Python Programing

Functions

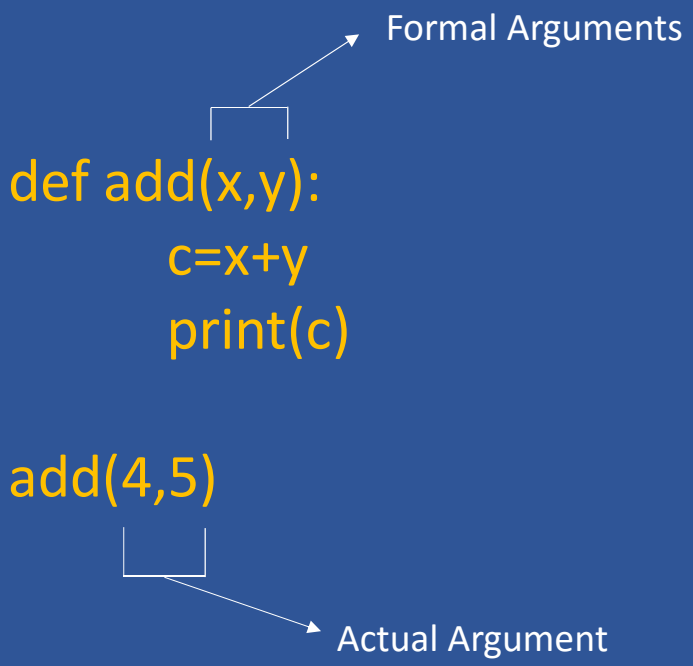
Function Definition

```
def greet():
        print("Hello")
        print("Good Morning")
```

Python Programing

Functions

Passing Parameter to function



Python Programing

Functions

Types of arguments

- Position
- Keyword
- Default
- Variable length

Functions

Position Argument

```
def person(name,age):
        print(name)
        print(age)

person(“Harminder”,29)
```

Functions

Keyword Argument

```
def person(name,age):
        print(name)
        print(age)
person(age=11,name="Harminder")
```

Functions

Default Argument

```
def person(name,age):
        print(name)
        print(age)
person(age=11,name="Harminder")
```

Functions

Variable Length Argument

```
def sum(a,*b):
        for i in b:
                c=a+i
                print(c)

sum(2,4,6)
```

Functions

Keyworded Variable Length Argument

```
def person(a,**b):
        print(a)
        print(b)

person("Harminder",city="faridabad",age=19)
```

Functions

Keyworded Variable Length Argument

```
def person(a,**b):
        print(a)
        for i,j in b.items():
                print(i,j)

person("Harminder",city="faridabad",age=19)
```

Functions

Returning values from function

```
def add(x,y):
        c=x+y
        return c

a=add(4,5)
print(a)
```

Functions

Returning multiple values from function

```
def add_sub(x,y):
        c=x+y
        d=x-y
        return c,d

a,b=add_sub(4,5)
print(a,b)
```

Python Programing

Functions

Global & Local Variables

```
a=10
def hello():
        a=15
        print(a)

hello()

print(a)
```

Python Programing

Functions

Local Variables can only be used inside function

```
def hello():
        a=15
        print(a)

print(a)
```

This code will generate a Error

Functions

Global Variables can be used anywhere in Program

```
a=10
def hello():
        print(a)

hello()
```

Functions

Changing Value of a global Variable

```
a=10
def hello():
        global a
        a=15
        print(a)

hello()
print(a)
```

Functions

Passing List/tuple/set to a function

```
def hello(a,b,c):
        print(a)
        print(b)
        print(c)

a=[1,2,3,4,5]
b=(1,2,3,4,5)
c={1,2,3,4,5}

hello(a,b,c)
```

Functions

Anonymous Function(LAMBDA)

```
f= lambda a,b:a+b

result = f(5,6)

print(result)
```

Functions

Using Filter with lambda

```
nums = [2,3,45,6,7,8,80]

r= filter(lambda n:n%2==0,nums)

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

Functions

Using Map with lambda

```
nums = [2,3,45,6,7,8,80]

r= map(lambda n:n*2,nums)

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

Functions

Using Reduce with lambda

```
from functools import reduce
nums = [2,3,45,6,7,8,80]

r= reduce(lambda a,b:a+b,nums)

print(r)
```

Functions

Creating Modules in Python

```
def add(a,b):
        c=a+b
        return c

def sub(a,b):
        c=a-b
        return c

def mul(a,b):
        c=a*b
        return c
```

Python Programing

Functions

Using User Defined Modules in Python

```
import hello as h

r=h.add(3,4)

print(r)
```

Special Variable

Python Programing

```
import hi as h

def fun1():
        print("This is function 1")
        h.add()

def fun2():
        print("This is function 2")

def main():
        fun1()
        fun2()

main()
```

```
def add():
        print("This is add function")

def mul():
        print("This is mul function")

def hello():
        add()
        mul()

if __name__ == "__main__":
        hello()
```

Python Programing

Object Oriented Programing

Python Programing

OOP

Class Definition

```
class a:
        def add(self):
                print(“This is add function")

obj = a()
a.add(obj)
obj.add()
```

Python Programing

OOP

Multiple Object of a class

```
class a:
        def add(self):
                print(“This is add function")

obj = a()
obj2=a()
obj.add()
obj2.add()
```

OOP

__init__ Method

```
class a:
        def __init__(self)
                print(“This is init function")

obj = a()
```

OOP

Passing Arguments to a Method

```
class person:
        def a(self,name):
                print("Hi",name)

obj = person()
obj.a("Harminder")
```

OOP

Accessing Variable

```
class person:
        x=1

obj = person()
print(obj.x)
```

Binding variable to object

OOP

```
class a:
        def b(self,k=5,n=4):
                self.k=k
                self.n=n
        def c(self):
                print(self.k,self.n)


obj = a()
obj.b(44,67)
obj.c()

obj2 = a()
obj2.b()
obj2.c()
```

Binding variable to object using __init__

OOP

```
class a:
        def __init__(self,k=5,n=4):
                self.k=k
                self.n=n
        def c(self):
                print(self.k,self.n)

obj = a(44,67)
obj.c()

obj2 = a()
obj2.c()
```

Instance Variable

OOP

```
class a:
        def __init__(self):
                self.b=5

c1=a()
c2=a()
print(c1.b)
print(c2.b)
c1.b=10
print(c1.b)
print(c2.b)
```

Class Variable

OOP

```
class a:
        x=4
        def __init__(self):
                self.b=5

c1=a()
c2=a()
print(c1.x)
print(c2.x)
a.x=15
print(c1.x)
print(c2.x)

c1.x=55
print(c1.x)
print(c2.x)
```

Python Programing

Types of Methods

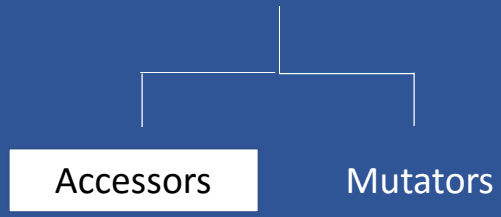
(Instance Methods)

OOP

```
class student:
        def __init__(self,m1,m2,m3):
                self.m1=m1
                self.m2=m2
                self.m3=m3

        def avg(self):
                return (self.m1+self.m2+self.m3)/3

s1 = student(23,56,44)
s2 = student(90,89,45)
print(s1.avg())
print(s2.avg())
```

Types of Methods

(Instance Methods)



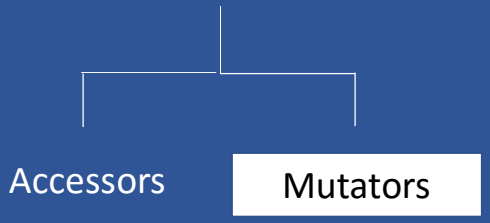
OOP

```
class student:
        def __init__(self):
                self.a="Harminder"

        def get_a(self):
                print(self.a)

s1 = student()
s1.get_a()
```

Python Programing

Types of Methods

(Instance Methods)



OOP

```
class student:
        def __init__(self):
                self.a="Harminder"

        def set_a(self):
                self.a="surender"
                return self.a

s1 = student()
print(s1.a)
print(s1.set_a())
```

Types of Methods
(Class Methods)

OOP

```
class student:
        school="vsics"

        @classmethod
        def get_school(cls):
                print(cls.school)

s1=student()
student.get_school()
```

Python Programing

Types of Methods
(Static Methods)

OOP

```
class student:
        @staticmethod
        def a():
                print("hi")

s1=student()
s1.a()
```

Python Programing

Inner Class

OOP

```
class student:
        def a(self):
                print("hi")
                self.obj = self.b()
                self.obj.hello()

        class b:
                def hello(self):
                        print("hello")

s1=student()
s1.a()
```

Inner Class

(using Object of inner class outside main class)

OOP

```
class student:
        def a(self):
                print("hi")
                self.obj = self.b()

        class b:
                def hello(self):
                        print("hello")

s1=student()
s1.a()
s1.obj.hello()
```

Inner Class

(Defining Object of inner class outside main class)

OOP

```
class student:
        def a(self):
                print("hi")

        class b:
                def hello(self):
                        print("hello")

s1=student()
obj=s1.b()
obj.hello()
```

Inheritance

OOP

```
class a:
        def feature1(self):
                print("Feature 1 is working")

        def feature2(self):
                print("Feature 2 is working")

class b(a):
        def feature3(self):
                print("Feature 3 is working")

        def feature4(self):
                print("Feature 4 is working")

obj1 = b()
obj1.feature1()
```

Multi Level Inheritance

OOP

```
class a:
        def feature1(self):
                print("Feature 1 is working")

        def feature2(self):
                print("Feature 2 is working")

class b(a):
        def feature3(self):
                print("Feature 3 is working")

        def feature4(self):
                print("Feature 4 is working")

class c(b):
        def feature5(self):
                print("Feature 5 is working")

obj1 = c()
obj1.feature1()
```

Multiple Inheritance

OOP

```
class a:
        def feature1(self):
                print("Feature 1 is working")

        def feature2(self):
                print("Feature 2 is working")

class b:
        def feature3(self):
                print("Feature 3 is working")

        def feature4(self):
                print("Feature 4 is working")

class c(a,b):
        def feature5(self):
                print("Feature 5 is working")

obj1 = c()
obj1.feature1()
```

Constructor Behavior in Single/Multi level inheritance

OOP

```
class a:
        def __init__(self):
                print("Init of a")
        def feature1(self):
                print("This is feature 1")
        def feature2(self):
                print("This is Feature 2")

class b(a):
        def __init__(self):
                super().__init__()
                print("Init of b")
        def feature3(self):
                print("This is feature 3")
        def feature4(self):
                print("This is Feature 4")

k = b()
```

Python Programing

Constructor Behavior in Multiple Inheritance (MRO)

OOP

```
class a:
        def __init__(self):
                super().__init__()
                print("Init of a")
        def feature1(self):
                print("This is feature 1")
        def feature2(self):
                print("This is Feature 2")

class b:
        def __init__(self):
                super().__init__()
                print("Init of b")
        def feature3(self):
                print("This is feature 3")
        def feature4(self):
                print("This is Feature 4")

class c(a,b):
        def __init__(self):
                super().__init__()
                print("Init of c")
        def feat(self):
                print("This is feat")

k = c()
```

OOP

Polymorphism

- Duck Typing
- Operator Overloading
- Method Overriding

Python Programing

Using methods in other classes

OOP

```
class b:
        def k(self):
                print("This is k function")

class a:
        def a(self,obj2):
                obj2.k()

obj2=b()
obj = a()
obj.a(obj2)
```

Duck Typing

OOP

```
class b:
        def k(self):
                print("This is k function")

class a:
        def a(self,obj2):
                obj2.k()

class d:
        def k(self):
                print("This is k in d")


obj2=d()
obj = a()
obj.a(obj2)
```

OOP

Operator Overloading

Python Programing

OOP

Operator Overloading

(Everything in python is a class)

```
a=4
b=5
c=a+b
print(c)

print(int.__add__(a,b))
```

Python Programing

OOP

Operator Overloading
(Int class has various methods)

+	• __add__()
-	• __sub__()
*	• __mul__()

OOP

Overloading Addition Operator

```
class a:
        def __init__(self,m1,m2):
                self.m1=m1
                self.m2=m2
        def __add__(obj1,obj2):
                x = obj1.m1+obj2.m1
                y = obj1.m2+obj2.m2
                z = a(x,y)
                return z


s1 = a(3,4)
s2 = a(44,55)

s3 = s1+s2
print(s3.m1)
```

Python Programing

OOP

Overloading Greater than Operator

```
class a:
        def __init__(self,m1,m2):
                self.m1=m1
                self.m2=m2
        def __gt__(obj1,obj2):
                x = obj1.m1+obj1.m2
                y = obj2.m1+obj2.m2
                if x>y:
                        return True
                else:
                        return False

s1 = a(3,4)
s2 = a(44,55)

if s1>s2:
        print("s1 wins")
else:
        print("s2 wins")
```

Python Programing

OOP

Method Overriding

```
class a:
        def greet(self):
                print("Welcome to class a")

class b(a):
        def greet(self):
                print("Welcome to class b")

obj = b()
obj.greet()
```

OOP

Iterator

```
a = [2,33,45,67,890,3]

c = iter(a)

print(c.__next__())

for i in a:
        print(c.__next__())
```

Python Programing

OOP

Generators Example 1

```
def hello():
        yield 1
        yield 2
        yield 3

values=hello()
print(values.__next__())
print(values.__next__())
print(values.__next__())
```

OOP

Generators Example 2

```
def sq():
        n=1
        while n<=10:
                yield n*n
                n+=1

values=sq()

print(next(values))
for i in values:
        print(i)
```

Python Programing

Exception Handling

Python Programing

Exception Handling

Types of Errors

- Compile Time
- Logical
- Runtime Error

Python Programing

Exception Handling

Types of Statements

a=25
b=5

Normal Statement

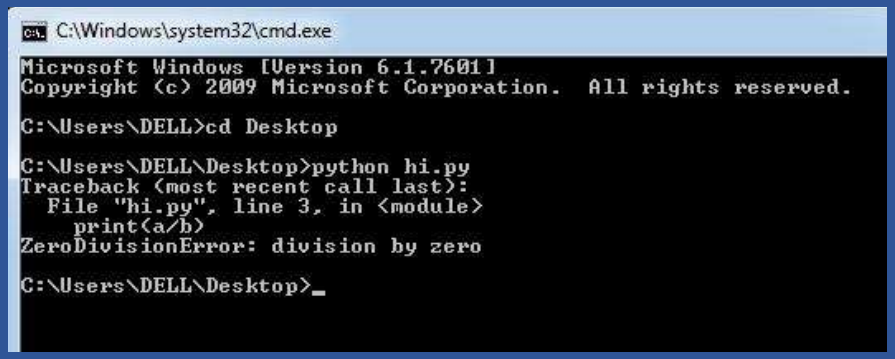
c=a/b → Critical Statement

Python Programing

Exception Handling

Runtime Error Example

```
a=25
b=0
print(a/b)
```



Exception Handling

Runtime Error Example

```
a=5
b=0

try:
        print(a/b)
except Exception:
        print("You cannot divide a number by zero")

print("bye")
```

Exception Handling

Try/Except/finally

```
a=5
b=2

try:
        print("Calculation mode started")
        print(a/b)

except Exception:
        print("You cannot divide a number by
zero")

finally:
        print("Calculation mode closed")
```

Exception Handling

Handling Specific Errors

```
a=5
b=2

try:
        print("Calculation mode started")
        print(a/b)

except ZeroDivisonError:
        print("You cannot divide a number by
zero")

finally:
        print("Calculation mode closed")
```

Multi Threading

Python Programing

Multi Threading

Multi Threading

```
from threading import *
from time import sleep
class hello(Thread):
        def run(self):
                for i in range(0,50):
                        print("hello")
                        sleep(1)

class hi(Thread):
        def run(self):
                for i in range(0,50):
                        print("hi")
                        sleep(1)

t1=hello()
t2=hi()

t1.start()
sleep(0.2)
t2.start()
```

Concept of Join

Multi Threading

```
from threading import *
from time import sleep
class hello(Thread):
        def run(self):
                for i in range(0,10):
                        print("hello")
                        sleep(1)

class hi(Thread):
        def run(self):
                for i in range(0,10):
                        print("hi")
                        sleep(1)

t1=hello()
t2=hi()

t1.start()
sleep(0.2)
t2.start()

t1.join()
print("bye")
```

File Handling

Python Programing

File Handling

Opening a file in Python

open(“filename”,”mode”)

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

File Handling

Reading Complete file

```
f=open("hello.txt","r")
print(f.read())
```

File Handling

Reading bits of a file

```
f=open("hello.txt","r")
print(f.read(6))
```

Python Programing

File Handling

Reading one line at a time

```
f=open("hello.txt","r")
print(f.readline())
print(f.readline())
```

File Handling

Reading Bits of a line

```
f=open("hello.txt","r")
print(f.readline())
print(f.readline(4))
```

File Handling

Writing a file

```
f=open("hello.txt","w")
f.write("hi who are
you??")
```

File Handling

Append to a file

```
f=open("hello.txt",“a")
f.write("hi who are
you??")
```

File Handling

Using for loop with file handler

```
f=open("hello.txt","r")

f1=open("hi.txt","a")

for i in f:
        f1.write(i)
```

File Handling

Removing a file

```
import os
os.remove("hi.txt")
```

File Handling

Removing a file

```
import os

if os.path.exists("hello.txt"):
        os.remove("hello.txt")
else:
        print("no file")
```

DJANGO

Python Programing

What is Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Python Programing

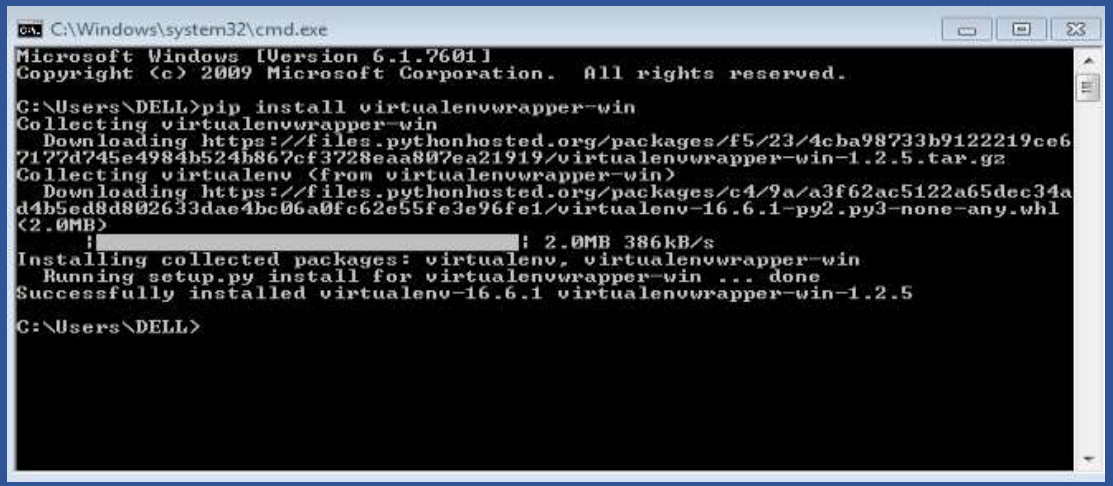
Why Django

- Fast
- Secure
- Scalable

Installation

Firstly we will install virtual environment wrapper

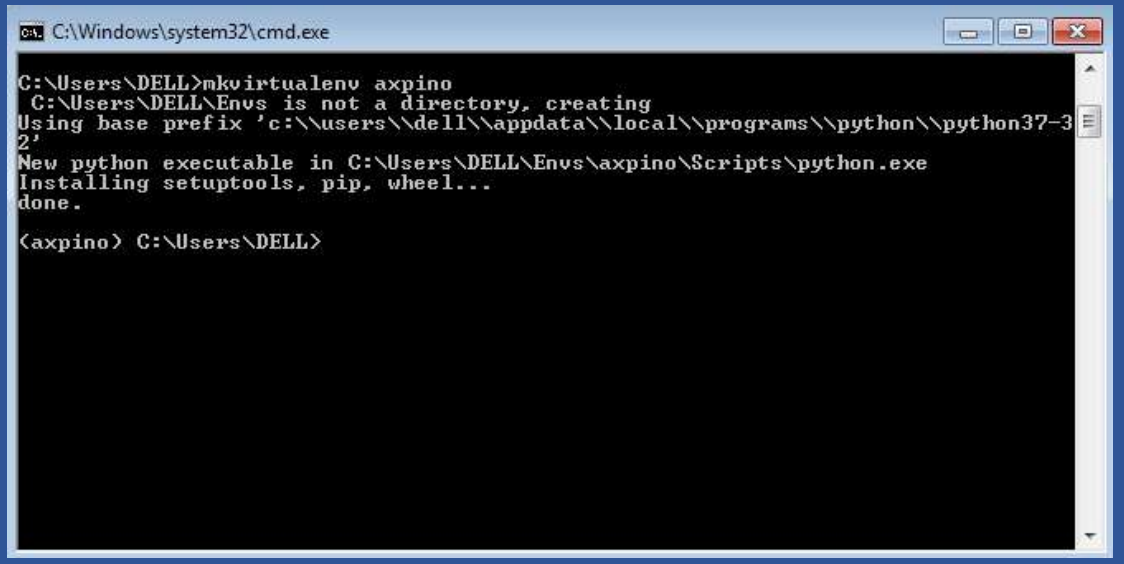
pip install virtualenvwrapper-win



Installation

Firstly we will create a virtual environment for Django

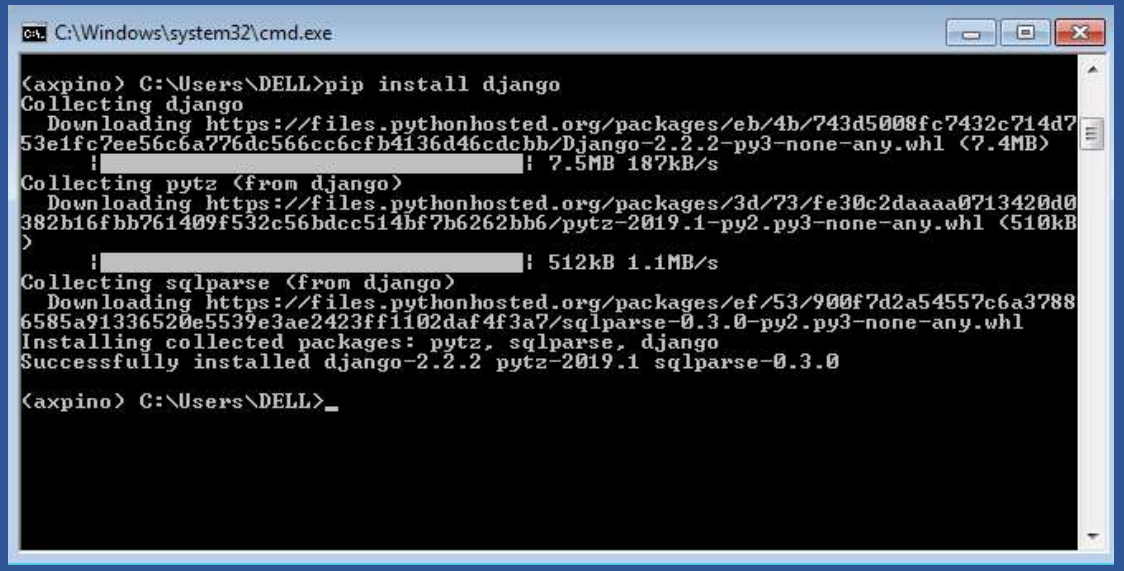
mkvirtualenv axpino



Installation

Now we will install Django into our Environment

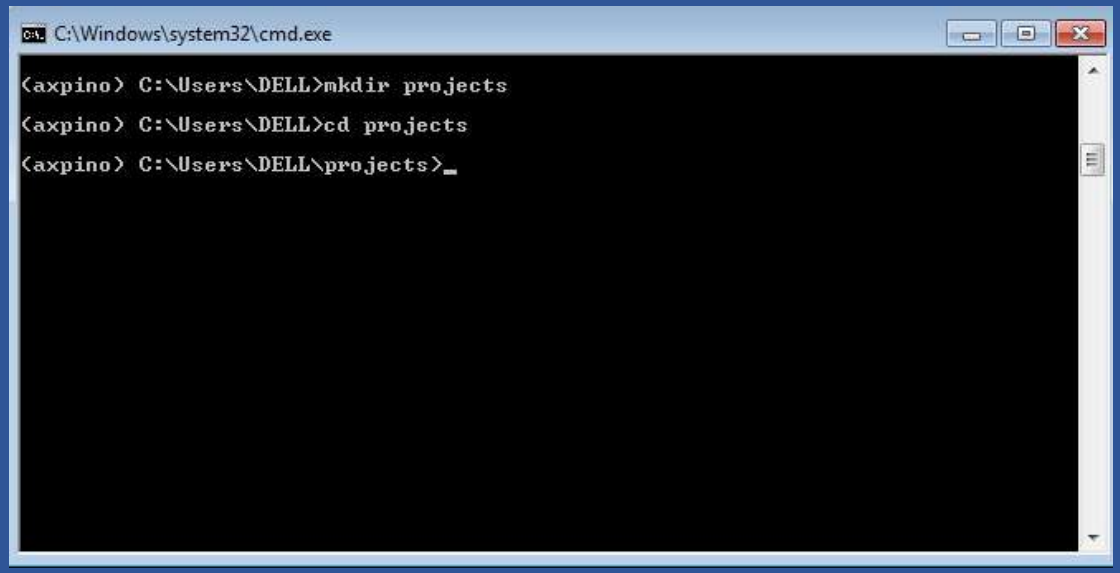
pip install django



Installation

Now we will create a folder to store our projects and navigate to it

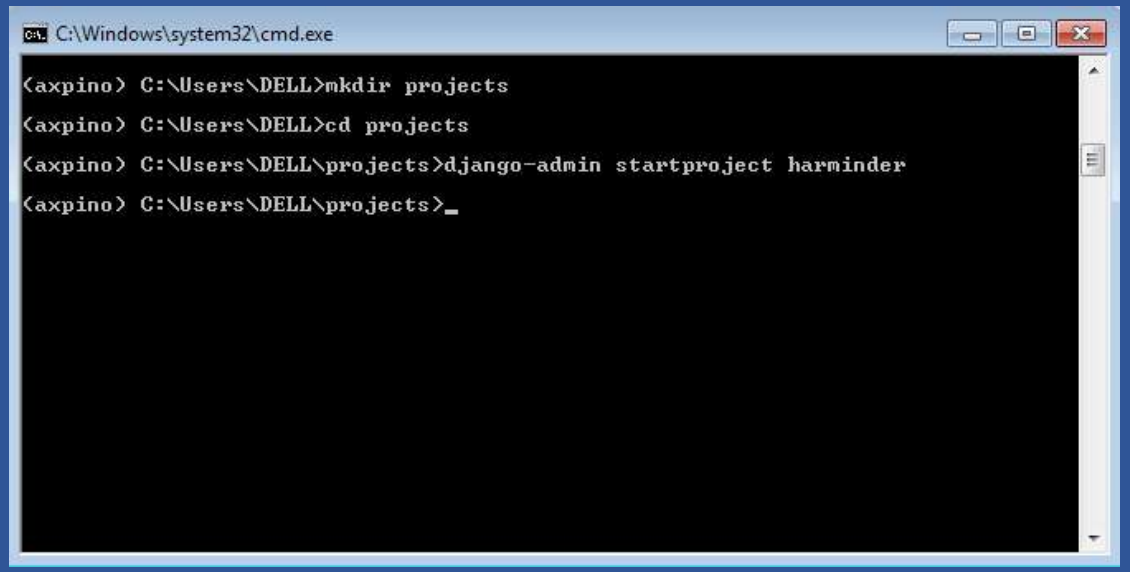
mkdir projects
cd projects



Installation

Now we will create our first project

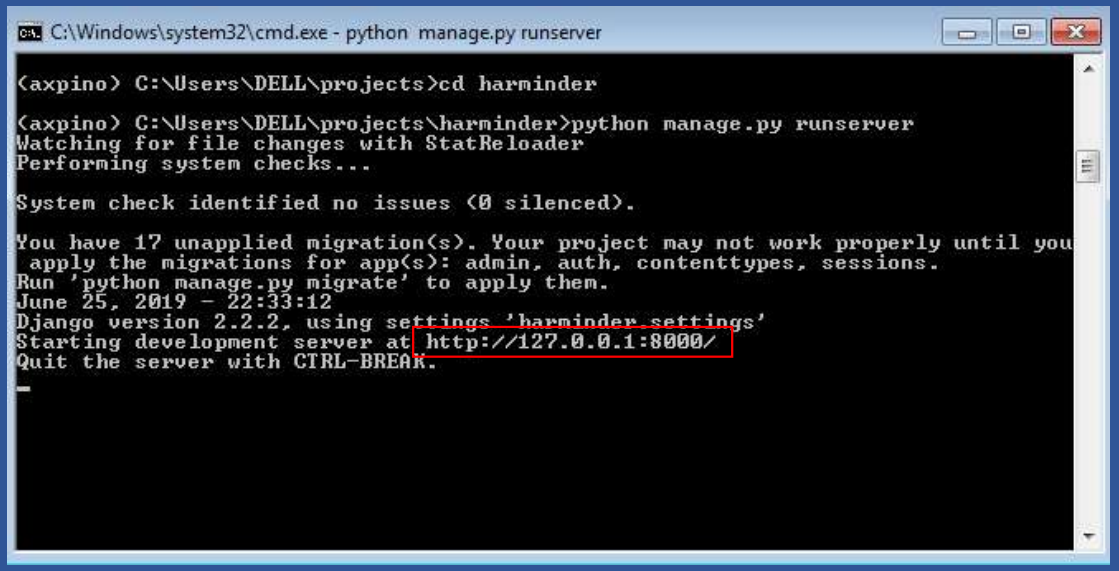
django-admin startproject projectname



Installation

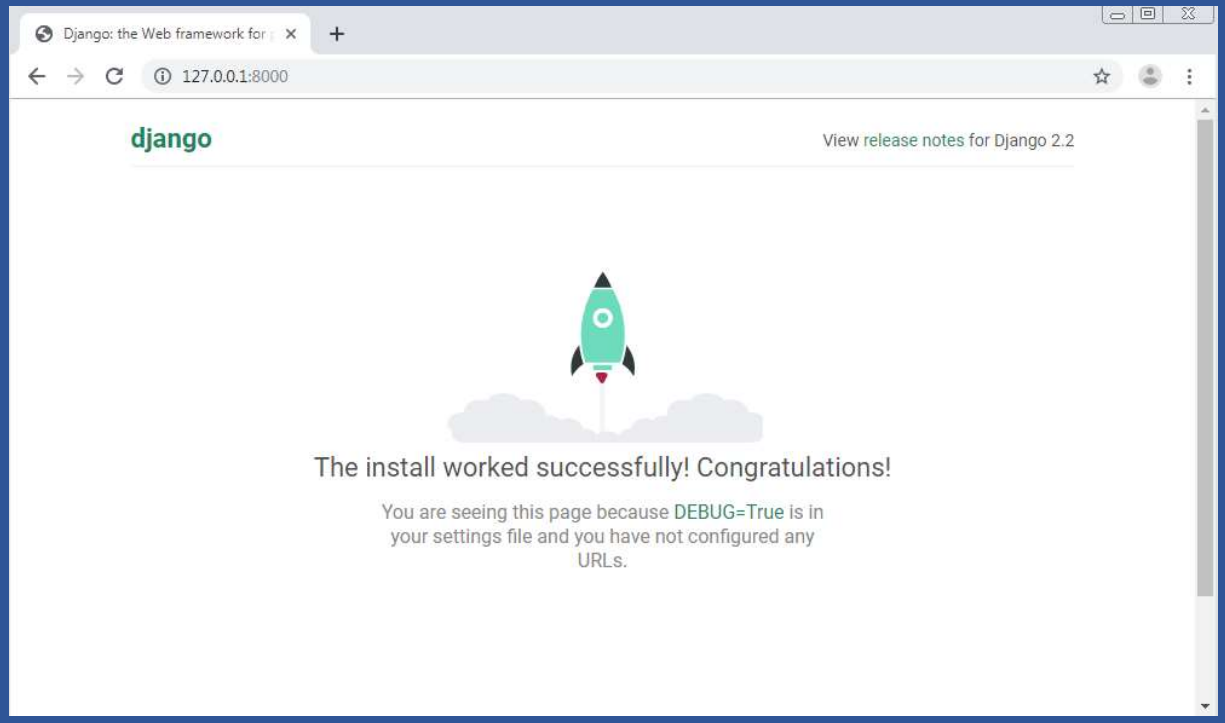
Navigate to project folder and start server

```
cd projectname
python manage.py runserver
```



Installation

Now lets access the main url of project



Creating views

Firstly we will create a app inside our project

Select the Environment using command

>>workon env_name

Navigate to projects folder

>>cd projects

Navigate to your project

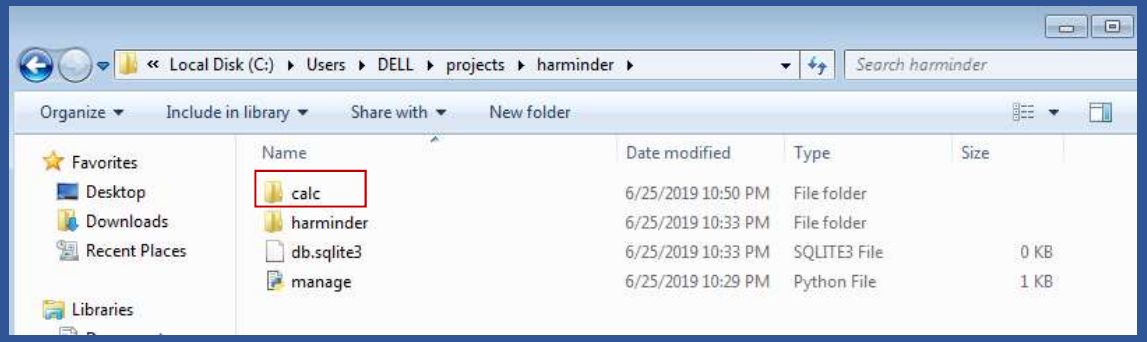
>>cd project_name

Create a app using command

>>python manage.py startapp app_name

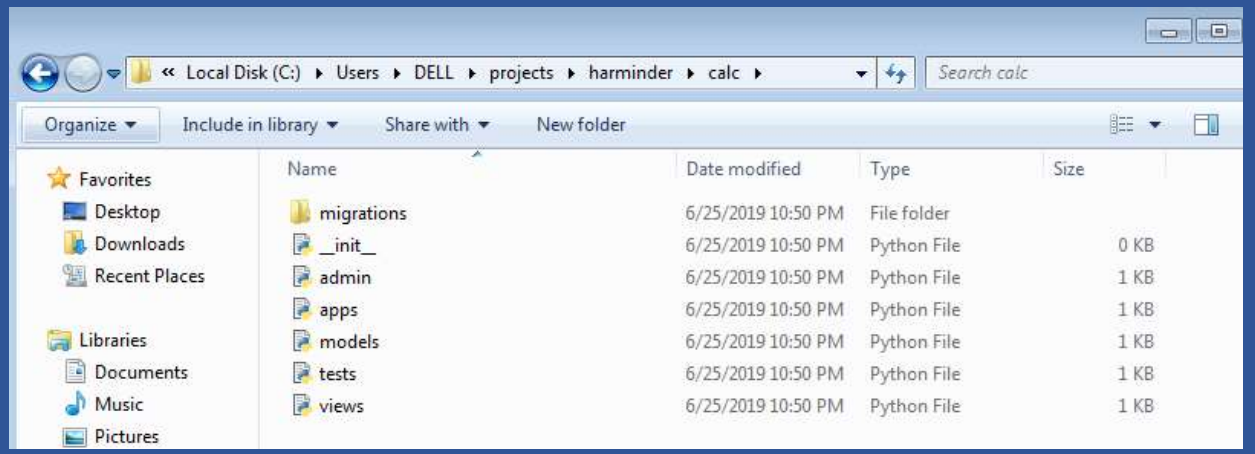
Creating views

Lets check our app folder



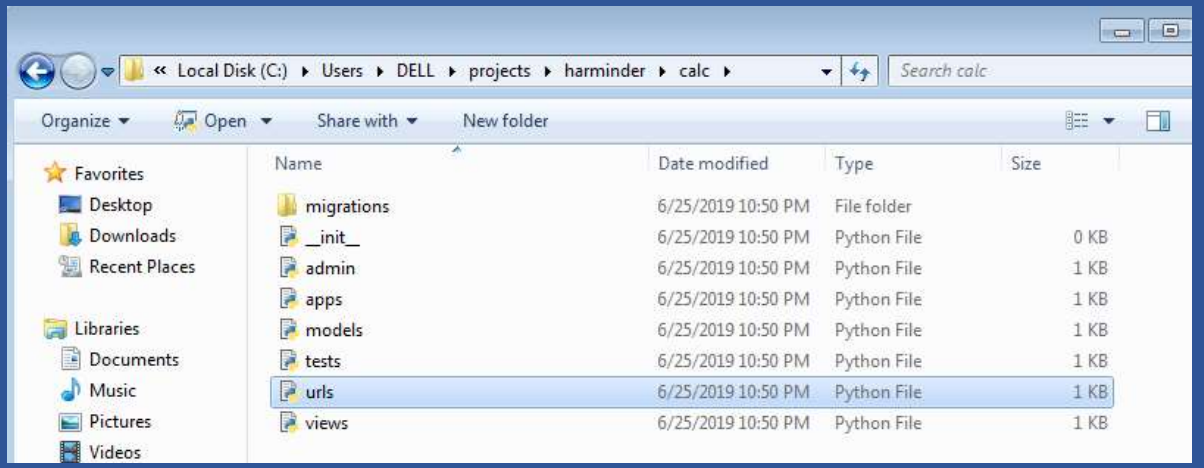
Creating views

Lets check our app folder



Creating views

To create a url we need to create a file with name urls in our app folder



Creating views

Let us add a path in urls file in our app

```
from django.urls import path
from . import views

urlpatterns = [
        path('',views.home,name="home")
]
```

Creating views

Let create a view function

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
def home(request):
        return HttpResponse("This is my webpage")
```

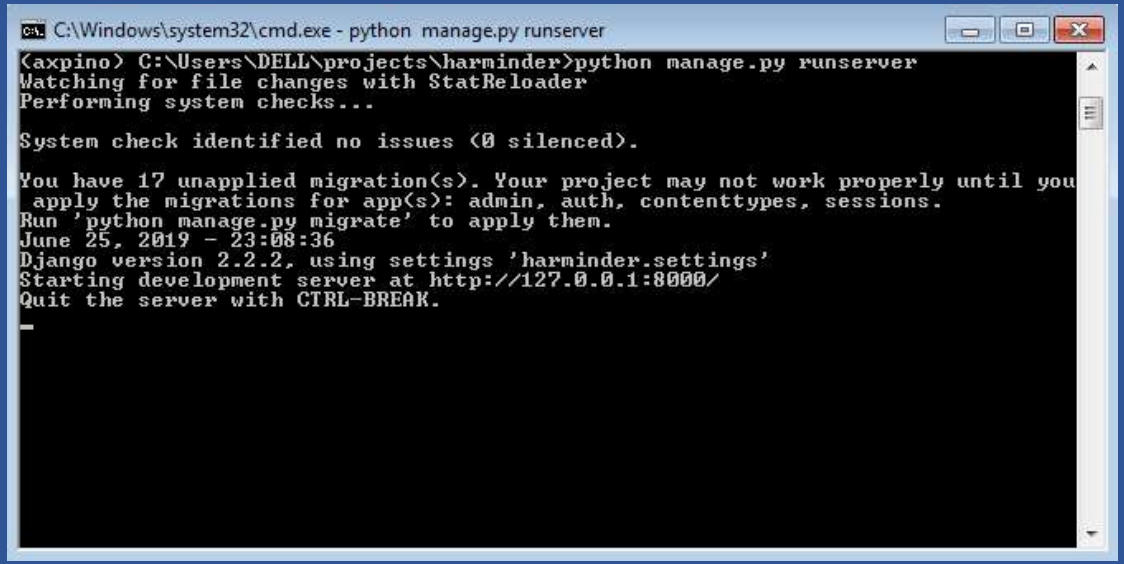
Creating views

Let add path in main project url

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
        path('',include('calc.urls')),
    path('admin/', admin.site.urls),
]
```

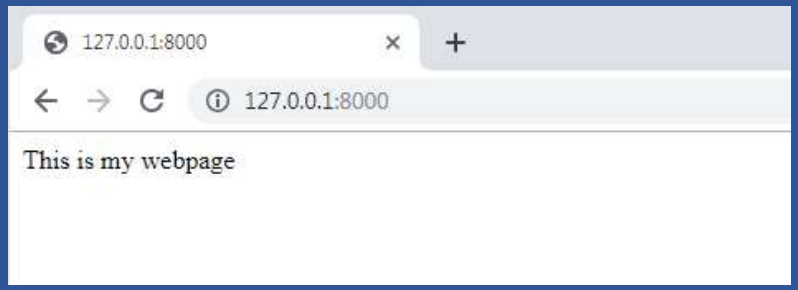
Creating views

Restart the Server



Creating views

Access the project using browser



Python Programing