**Lecture 29**

Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" f>>> import random

>>> import function\_Demo

>>> function\_Demo.is\_even(34)

'Even'

===================RESTART: Shell ===============================

>>> import function\_Demo as fd

>>> fd.is\_even("hl4k5")

'Not allowed'

>>> def power(a,b):

return a\*\*b

>>> power(2,3)

8

>>> power(3)

Traceback (most recent call last):

File "<pyshell#12>", line 1, in <module>

power(3)

TypeError: power() missing 1 required positional argument: 'b'

>>> #Default

>>> def power(a=1,b=1):

return a\*\*b

>>> # Positiona Argument

>>> power(3)

3

>>> power()

1

>>> power(3,3)

27

#keyword argument

>>> power(b=2,a=3)

9

>>> print(1,2,3)

1 2 3

>>> def flexi(\*number)

SyntaxError: invalid syntax

>>> def flexi(\*number)

SyntaxError: invalid syntax

>>> def flexi(\*number):

product =1

for i in number:

product = product \* i

print(product)

>>> flexi(1)

1

>>> flexi(1,2)

2

>>> flexi(1,2,3,4)

24

>>> def flexi(\*number):

product =1

print(number)

print(type(number))

for i in number:

product = product \* i

print(product)

>>> flexi(1,2,3,4)

(1, 2, 3, 4)

<class 'tuple'>

24

**>>> def f():**

print("Inside f")

def g():

print ("Inside g")

g()

>>> f()

Inside f

Inside g

>>> g()

Traceback (most recent call last):

File "<pyshell#48>", line 1, in <module>

g()

NameError: name 'g' is not defined

**>>> def f():**

print("Inside f")

def g():

print("Inside g")

f()

g()

infinite loop

**>>> # Functions as objects**

>>> def f(num):

return num\*\*2

>>> f(2)

4

>>> f(4)

16

>>> x = f

>>> x(4)

16

>>> x(2)

4

>>> del f

>>> x(2)

4

>>> type(x)

<class 'function'>

>>> L = [1,2,3,4]

>>> L

[1, 2, 3, 4]

>>> L = [1,2,3,4,x]

>>> L

[1, 2, 3, 4, <function f at 0x0000024D13E6C268>]

>>> L[-1](3)

9

>>> L =[1,2,3,4,x(5)]

>>> L

[1, 2, 3, 4, 25]

**Recursion**

def multiply(a,b):

result=0

for i in range(b):

result = result + a

print(result)

multiply(3,4)

>>> import Recursion

12

def mult(a,b):

if b==1:

return a

else:

return a + mult(a,b-1)

print(mult(4,4))

>>> import Recursion

16

**def fact(a):**

if a==1:

return 1

else:

return a \* fact(a-1)

print(fact(5))

>>> import Recursion

120

**def palin(s):**

if len(s)==1:

print("Palindrome")

else:

if s[0]== s[-1]:

palin(s[1:-1])

else:

print("Not a palindrome")

palin("madam")

palin("mama")

>>> import Recursion

Palindrome

Not a palindrome

**def fib(m):**

if m==0:

return 0

elif m==1:

return 1

else:

return fib(m-1)+fib(m-2)

print(fib(12))

>>> import Recursion

233

**LECTURE 31**

**Lambda Functions in Python**

>>> x = lambda x:x\*\*2

>>> x(9)

81

>>> a=lambda x,y:x+y

>>> a(4,5)

9

>>> #Difference

>>> # 1. Lambda has no return value

>>> type(a)

<class 'function'>

>>> #2. one line

>>> #3. Not useed for code reusability

>>> #4. No name

>>>

>>> #why?

>>> # Along with Higher order functions

>>>

>>> b = lambda x:x[0]== 'a'

>>> b('applw')

True

>>> b('banana')

False

>>>

>>> b = lambda x:"Even" if x%2==0 else "odd"

>>> b(3)

'odd'

>>> b(2)

'Even'

**Higher order function**

def return\_sum(func,L):

result=0

for i in L:

if func(i):

result = result + i

return result

L =[11,14,21,23,56,78,45,29,28]

x=lambda x:x%2==0

y=lambda x:x%2!=0

z= lambda x:x%3==0

print(return\_sum(x,L))

print(return\_sum(y,L))

print(return\_sum(z,L))

**>>> #Map**

>>> L =[1,2,3,4,5,6,7]

>>> L

[1, 2, 3, 4, 5, 6, 7]

>>> map(lambda x:x\*2,L)

<map object at 0x000001E2931BADA0>

>>> list(map(lambda x:x\*2,L))

[2, 4, 6, 8, 10, 12, 14]

>>>

>>>

>>> list(map(lambda x:x%2==0,L))

[False, True, False, True, False, True, False]

>>> students = [

{

"name" : "Jacob Martin",

"father name" : "Ros Martin",

"Address" : "123 Hill Street",

},{

"name" : "Angela Stevens",

"father name" : "Robert Stevens",

"Address" : "3 Upper Street London",

},{

"name" : "Ricky Smart",

"father name" : "William Smart",

"Address" : "Unknown",

}

]

>>> map(lambda student:student,L)

<map object at 0x000001E2930B5588>

>>> list(map(lambda student:student['name'],students))

['Jacob Martin', 'Angela Stevens', 'Ricky Smart']

>>> # Filter

>>> L

[1, 2, 3, 4, 5, 6, 7]

>>> list(filter(lambda x:x>4,L))

[5, 6, 7]

>>> fruits =['Apple','Orange','Mango','Guava']

>>> list(filter(lambda fruit:'e' in fruit,fruits))

['Apple', 'Orange']

>>> #Reduce

>>> import functools

>>> L

[1, 2, 3, 4, 5, 6, 7]

>>> functools.reduce(lambda x,y:x+y,L)

28

>>> L1 = [12,34,56,11,21,58]

>>> functools.reduce(lambda x,y:x if x>y else y,L1)

58

>>> functools.reduce(lambda x,y:x if x<y else y,L1)

11

>>> L1 =[item \* 2 for item in L]

>>> L1

[2, 4, 6, 8, 10, 12, 14]

>>> L2=[i \*\* 2 for i in range(10)]

>>> L2

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> L3 = [i\*\*2 for i in range(10) if i%2!=0]

>>> L3

[1, 9, 25, 49, 81]

List Comprehension

>>> D1 ={"Name":"Ram","Gender":"Male","Age":30}

>>> D2={key:value for key,value in D1.items() if len(key)>3}

>>> D2

{'Name': 'Ram', 'Gender': 'Male'}

>>>

>>> L

[1, 2, 3, 4, 5, 6, 7]

>>> D3 = {item:item\*\*2 for item in L}

>>> D3

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49}

**Lecture 32**

**Object Oriented Programming**

>>> L=[1,2,3,4]

>>> L

[1, 2, 3, 4]

>>> Lupper()

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>

Lupper()

NameError: name 'Lupper' is not defined

>>> city = "Kolkata"

>>> city.append("a")

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module>

city.append("a")

AttributeError: 'str' object has no attribute 'append'

>>> a=3

>>> a.upper()

Traceback (most recent call last):

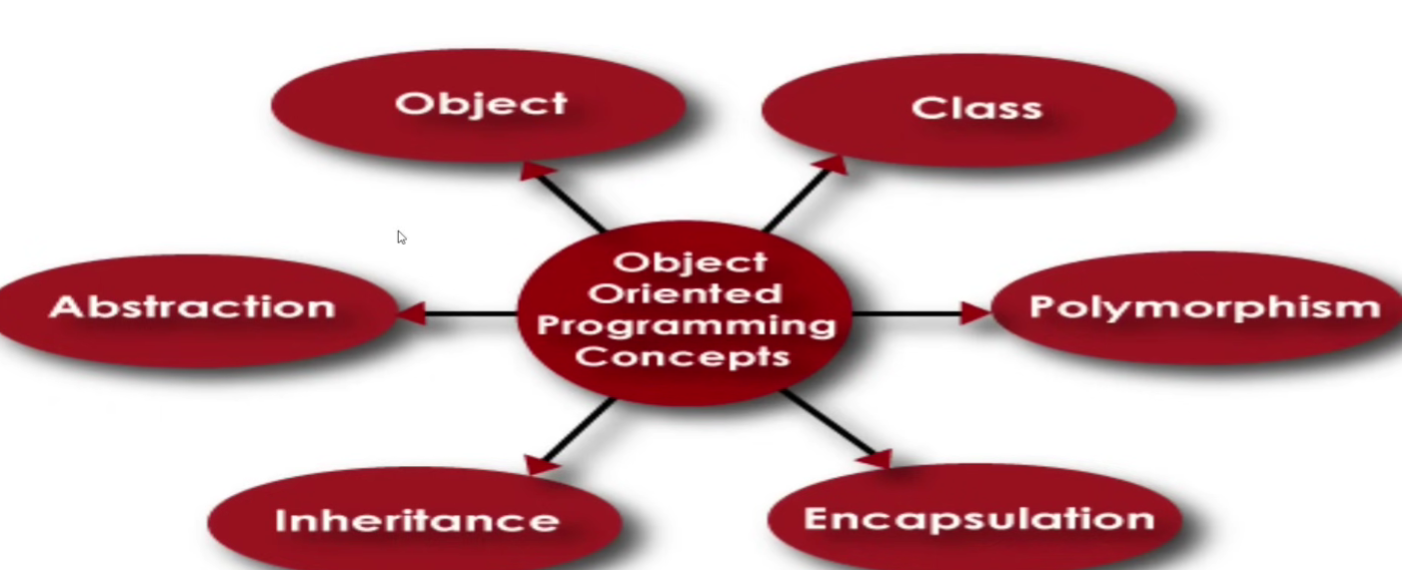
File "<pyshell#6>", line 1, in <module>

a.upper()

AttributeError: 'int' object has no attribute 'upper'

**Everything object Python**

Generality to Specification

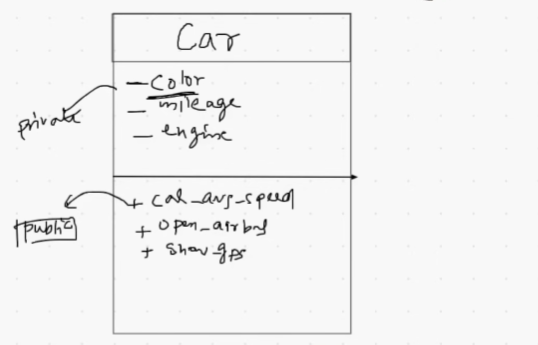
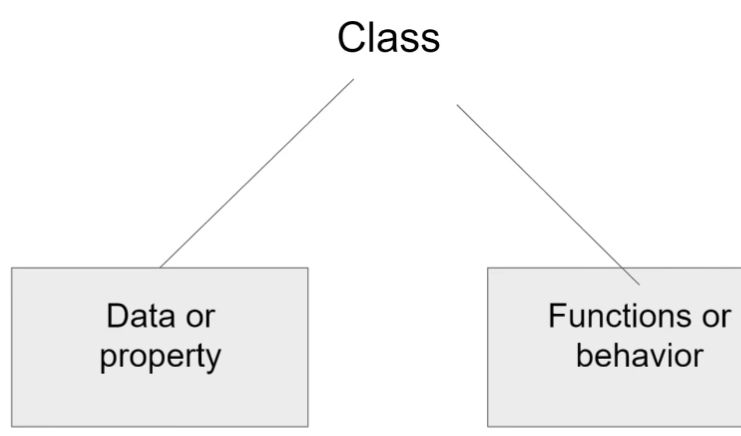


Class – is a blueprint

>>> a=2

>>> type(a)

<class 'int'>



>>> # Class name should be in Pascal case

>>> ThisIsPascalCase



>>> L=[1,2,3]

>>> L

[1, 2, 3]

>>> # Object Literal

>>>

>>> L = list()

>>> L

[]

>>> city = str()

>>> city

''

**Let us make a class**

#Function vs Methods

# Function inside class is called methods

>>> len(L) #function

0

Methods

>>> L.append(1)

>>> L

[1]

**Code1:**

1. The \_\_init\_\_ method is misspelled as \_init\_. It should have two underscores before and after the word ‘init’.

class Atm:

def \_\_init\_\_(self):

self.pin=""

self.balance=0

self.menu()

def menu(self):

user\_input=input("""

Hello, how would you like to proceed?

1. Enter 1 to create pin.

2. Enter 2 to deposit

3. Enter 3 to withdraw

4. Enter 4 to check balance

5. Enter 5 to exit

""")

if user\_input=="1":

print("create pin")

elif user\_input=="2":

print ("Withdraw")

elif user\_input=="3":

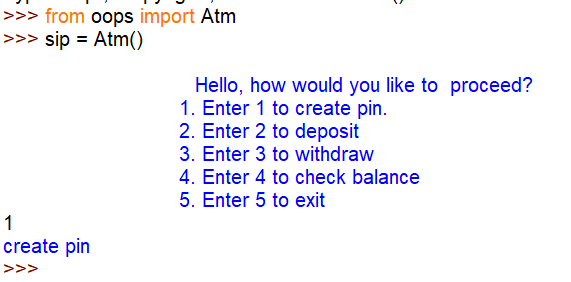
print("deposit")

elif user\_input=="4":

print("balance")

else:

print("bye")

 **Lecture 29**

Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" f>>> import random

>>> import function\_Demo

>>> function\_Demo.is\_even(34)

'Even'

===================RESTART: Shell ===============================

>>> import function\_Demo as fd

>>> fd.is\_even("hl4k5")

'Not allowed'

>>> def power(a,b):

return a\*\*b

>>> power(2,3)

8

>>> power(3)

Traceback (most recent call last):

File "<pyshell#12>", line 1, in <module>

power(3)

TypeError: power() missing 1 required positional argument: 'b'

>>> #Default

>>> def power(a=1,b=1):

return a\*\*b

>>> # Positiona Argument

>>> power(3)

3

>>> power()

1

>>> power(3,3)

27

#keyword argument

>>> power(b=2,a=3)

9

>>> print(1,2,3)

1 2 3

>>> def flexi(\*number)

SyntaxError: invalid syntax

>>> def flexi(\*number)

SyntaxError: invalid syntax

>>> def flexi(\*number):

product =1

for i in number:

product = product \* i

print(product)

>>> flexi(1)

1

>>> flexi(1,2)

2

>>> flexi(1,2,3,4)

24

>>> def flexi(\*number):

product =1

print(number)

print(type(number))

for i in number:

product = product \* i

print(product)

>>> flexi(1,2,3,4)

(1, 2, 3, 4)

<class 'tuple'>

24

**>>> def f():**

print("Inside f")

def g():

print ("Inside g")

g()

>>> f()

Inside f

Inside g

>>> g()

Traceback (most recent call last):

File "<pyshell#48>", line 1, in <module>

g()

NameError: name 'g' is not defined

**>>> def f():**

print("Inside f")

def g():

print("Inside g")

f()

g()

infinite loop

**>>> # Functions as objects**

>>> def f(num):

return num\*\*2

>>> f(2)

4

>>> f(4)

16

>>> x = f

>>> x(4)

16

>>> x(2)

4

>>> del f

>>> x(2)

4

>>> type(x)

<class 'function'>

>>> L = [1,2,3,4]

>>> L

[1, 2, 3, 4]

>>> L = [1,2,3,4,x]

>>> L

[1, 2, 3, 4, <function f at 0x0000024D13E6C268>]

>>> L[-1](3)

9

>>> L =[1,2,3,4,x(5)]

>>> L

[1, 2, 3, 4, 25]

**Recursion**

def multiply(a,b):

result=0

for i in range(b):

result = result + a

print(result)

multiply(3,4)

>>> import Recursion

12

def mult(a,b):

if b==1:

return a

else:

return a + mult(a,b-1)

print(mult(4,4))

>>> import Recursion

16

**def fact(a):**

if a==1:

return 1

else:

return a \* fact(a-1)

print(fact(5))

>>> import Recursion

120

**def palin(s):**

if len(s)==1:

print("Palindrome")

else:

if s[0]== s[-1]:

palin(s[1:-1])

else:

print("Not a palindrome")

palin("madam")

palin("mama")

>>> import Recursion

Palindrome

Not a palindrome

**def fib(m):**

if m==0:

return 0

elif m==1:

return 1

else:

return fib(m-1)+fib(m-2)

print(fib(12))

>>> import Recursion

233

**LECTURE 31**

**Lambda Functions in Python**

>>> x = lambda x:x\*\*2

>>> x(9)

81

>>> a=lambda x,y:x+y

>>> a(4,5)

9

>>> #Difference

>>> # 1. Lambda has no return value

>>> type(a)

<class 'function'>

>>> #2. one line

>>> #3. Not useed for code reusability

>>> #4. No name

>>>

>>> #why?

>>> # Along with Higher order functions

>>>

>>> b = lambda x:x[0]== 'a'

>>> b('applw')

True

>>> b('banana')

False

>>>

>>> b = lambda x:"Even" if x%2==0 else "odd"

>>> b(3)

'odd'

>>> b(2)

'Even'

**Higher order function**

def return\_sum(func,L):

result=0

for i in L:

if func(i):

result = result + i

return result

L =[11,14,21,23,56,78,45,29,28]

x=lambda x:x%2==0

y=lambda x:x%2!=0

z= lambda x:x%3==0

print(return\_sum(x,L))

print(return\_sum(y,L))

print(return\_sum(z,L))

**>>> #Map**

>>> L =[1,2,3,4,5,6,7]

>>> L

[1, 2, 3, 4, 5, 6, 7]

>>> map(lambda x:x\*2,L)

<map object at 0x000001E2931BADA0>

>>> list(map(lambda x:x\*2,L))

[2, 4, 6, 8, 10, 12, 14]

>>>

>>>

>>> list(map(lambda x:x%2==0,L))

[False, True, False, True, False, True, False]

>>> students = [

{

"name" : "Jacob Martin",

"father name" : "Ros Martin",

"Address" : "123 Hill Street",

},{

"name" : "Angela Stevens",

"father name" : "Robert Stevens",

"Address" : "3 Upper Street London",

},{

"name" : "Ricky Smart",

"father name" : "William Smart",

"Address" : "Unknown",

}

]

>>> map(lambda student:student,L)

<map object at 0x000001E2930B5588>

>>> list(map(lambda student:student['name'],students))

['Jacob Martin', 'Angela Stevens', 'Ricky Smart']

>>> # Filter

>>> L

[1, 2, 3, 4, 5, 6, 7]

>>> list(filter(lambda x:x>4,L))

[5, 6, 7]

>>> fruits =['Apple','Orange','Mango','Guava']

>>> list(filter(lambda fruit:'e' in fruit,fruits))

['Apple', 'Orange']

>>> #Reduce

>>> import functools

>>> L

[1, 2, 3, 4, 5, 6, 7]

>>> functools.reduce(lambda x,y:x+y,L)

28

>>> L1 = [12,34,56,11,21,58]

>>> functools.reduce(lambda x,y:x if x>y else y,L1)

58

>>> functools.reduce(lambda x,y:x if x<y else y,L1)

11

>>> L1 =[item \* 2 for item in L]

>>> L1

[2, 4, 6, 8, 10, 12, 14]

>>> L2=[i \*\* 2 for i in range(10)]

>>> L2

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> L3 = [i\*\*2 for i in range(10) if i%2!=0]

>>> L3

[1, 9, 25, 49, 81]

List Comprehension

>>> D1 ={"Name":"Ram","Gender":"Male","Age":30}

>>> D2={key:value for key,value in D1.items() if len(key)>3}

>>> D2

{'Name': 'Ram', 'Gender': 'Male'}

>>>

>>> L

[1, 2, 3, 4, 5, 6, 7]

>>> D3 = {item:item\*\*2 for item in L}

>>> D3

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49}

**Lecture 32**

**Object Oriented Programming**

>>> L=[1,2,3,4]

>>> L

[1, 2, 3, 4]

>>> Lupper()

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>

Lupper()

NameError: name 'Lupper' is not defined

>>> city = "Kolkata"

>>> city.append("a")

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module>

city.append("a")

AttributeError: 'str' object has no attribute 'append'

>>> a=3

>>> a.upper()

Traceback (most recent call last):

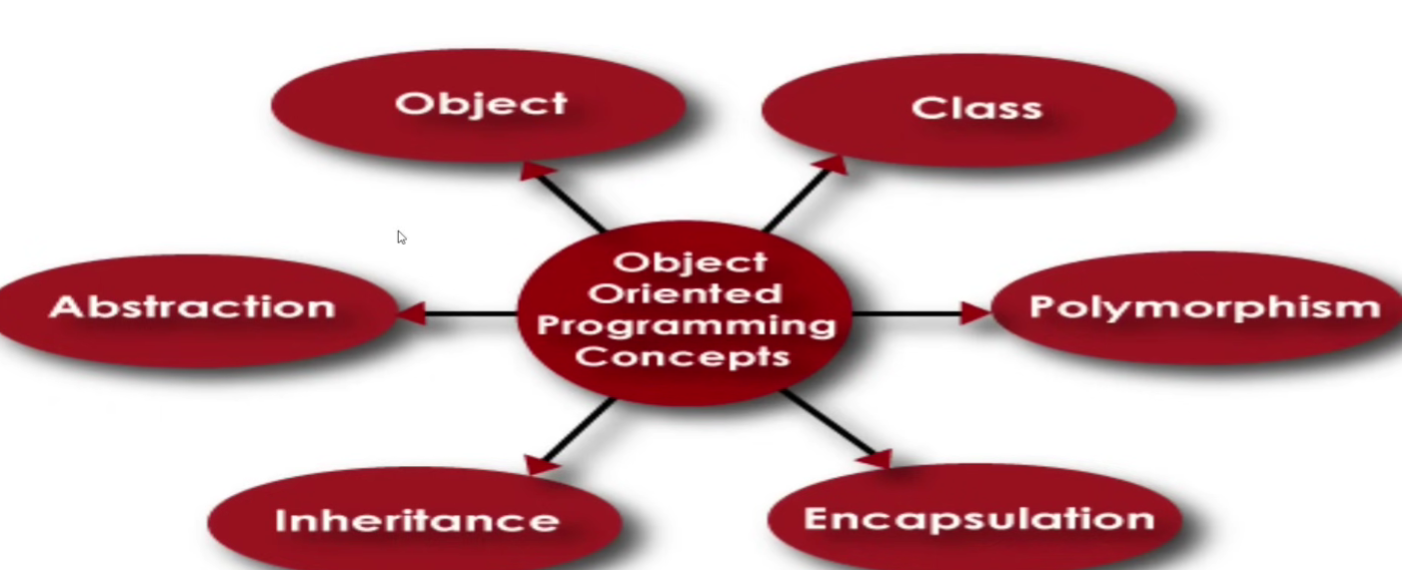
File "<pyshell#6>", line 1, in <module>

a.upper()

AttributeError: 'int' object has no attribute 'upper'

**Everything object Python**

Generality to Specification

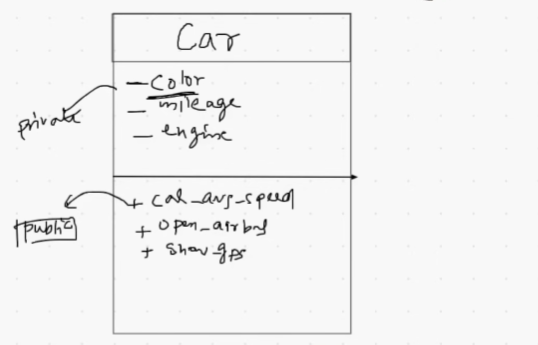
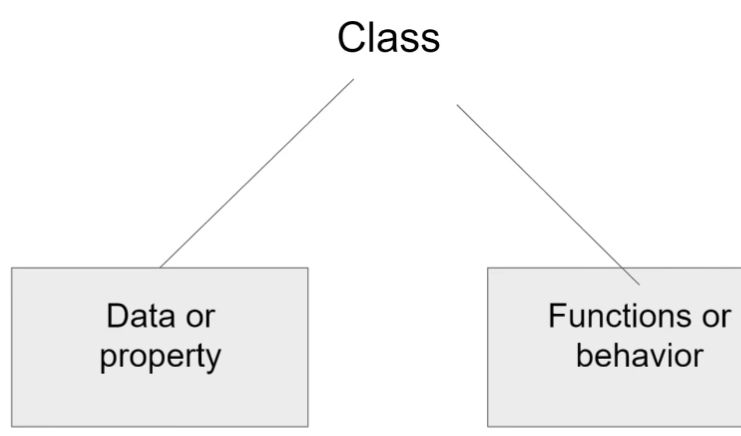


Class – is a blueprint

>>> a=2

>>> type(a)

<class 'int'>



>>> # Class name should be in Pascal case

>>> ThisIsPascalCase



>>> L=[1,2,3]

>>> L

[1, 2, 3]

>>> # Object Literal

>>>

>>> L = list()

>>> L

[]

>>> city = str()

>>> city

''

**Let us make a class**

#Function vs Methods

# Function inside class is called methods

>>> len(L) #function

0

Methods

>>> L.append(1)

>>> L

[1]

**Code1:**

1. The \_\_init\_\_ method is misspelled as \_init\_. It should have two underscores before and after the word ‘init’.

class Atm:

def \_\_init\_\_(self):

self.pin=""

self.balance=0

self.menu()

def menu(self):

user\_input=input("""

Hello, how would you like to proceed?

1. Enter 1 to create pin.

2. Enter 2 to deposit

3. Enter 3 to withdraw

4. Enter 4 to check balance

5. Enter 5 to exit

""")

if user\_input=="1":

print("create pin")

elif user\_input=="2":

print ("Withdraw")

elif user\_input=="3":

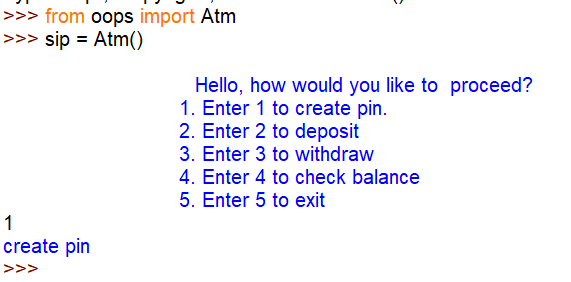
print("deposit")

elif user\_input=="4":

print("balance")

else:

print("bye")



class Atm:

def \_\_init\_\_(self):

self.pin=" "

self.balance=0

self.menu()

def menu(self):

user\_input=input("""

Hello, how would you like to proceed?

1. Enter 1 to create pin.

2. Enter 2 to deposit

3. Enter 3 to withdraw

4. Enter 4 to check balance

5. Enter 5 to exit

""")

if user\_input=="1":

self.create\_pin()

elif user\_input=="2":

self.deposit()

elif user\_input=="3":

self.withdraw()

elif user\_input=="4":

self.check\_balance()

else:

print("bye")

def create\_pin(self):

self.pin = input("Enter your pin")

print("Pin is set successfully")

def deposit(self):

temp = input("Enter your pin")

if temp == self.pin:

amount = int(input("Enter the amount"))

self.balance = self.balance + amount

print("Deposit successful")

else:

print("Invalid Pin")

def withdraw(self):

temp = input("Enter your pin")

if temp == self.pin:

amount = int(input("Enter the amount"))

if amount < self.balance:

self.balance = self.balance - amount

print("Operation successfully")

else:

print("insufficient funds")

else:

print("Invalid pin")

def check\_balance(self):

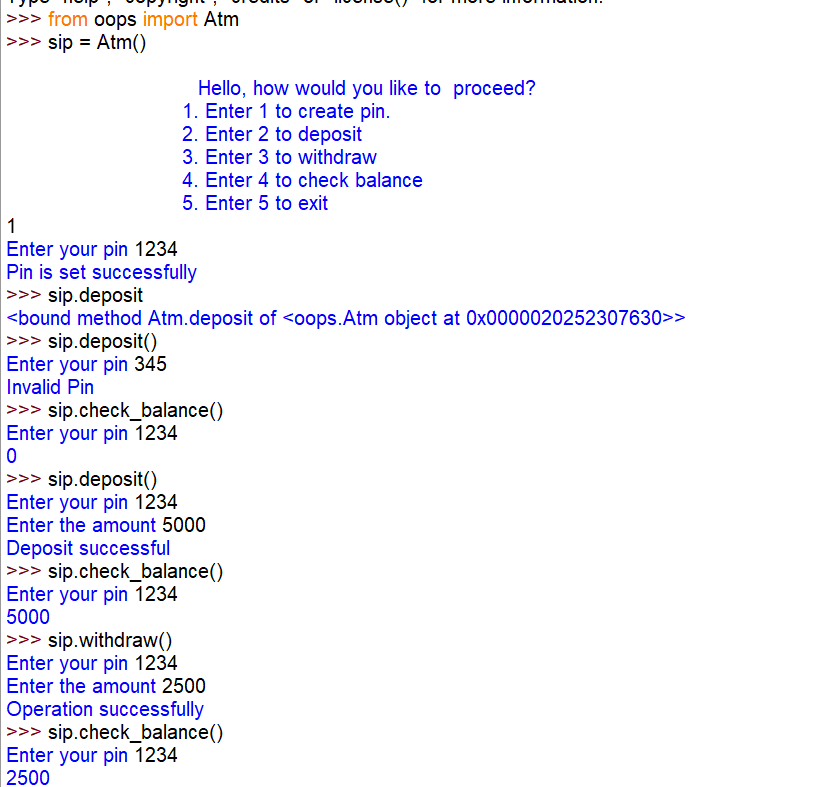
temp = input("Enter your pin")

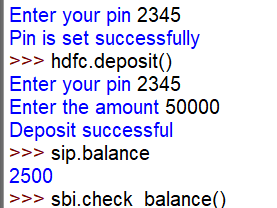
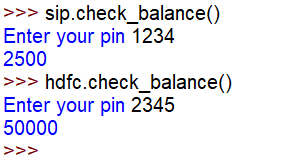
if temp == self.pin:

print(self.balance)

else:

print("invalid Pin")



# special/magic/dunder methods

**Self** - SELF represents the instance of class. This handy keyword allows you to access variables, attributes, and methods of a defined class in Python.

class Atm:

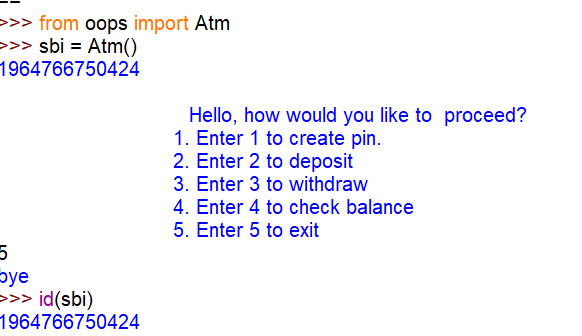
def \_\_init\_\_(self):

self.pin=" "

self.balance=0

print(id(self))

self.menu()



Currently we working with an object is called a Self.

https://tutorialsteacher.com/python/magic-methods-in-python

class Fraction:

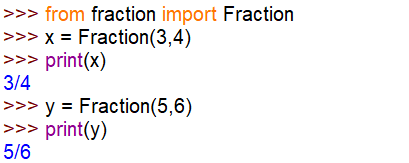
def \_\_init\_\_(self,n,d):

self.num = n

self.den=d

def \_\_str\_\_(self):

return "{}/{}".format(self.num,self.den)



class Fraction:

def \_\_init\_\_(self,n,d):

self.num = n

self.den=d

def \_\_str\_\_(self):

return "{}/{}".format(self.num,self.den)

def \_\_add\_\_(self,other):

temp\_num = self.num \* other.den + other.num \* self.den

temp\_den = self.den \* other.den

return "{}/{}".format(temp\_num,temp\_den)

def \_\_sub\_\_(self,other):

temp\_num = self.num \* other.den - other.num \* self.den

temp\_den = self.den \* other.den

return "{}/{}".format(temp\_num,temp\_den)

def \_\_mul\_\_(self,other):

temp\_num = self.num \* other.num

temp\_den = self.den \* other.den

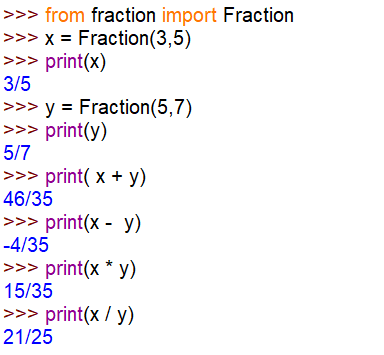
return "{}/{}".format(temp\_num,temp\_den)

def \_\_truediv\_\_(self,other):

temp\_num = self.num \* other.den

temp\_den = self.den \* other.num

return "{}/{}".format(temp\_num,temp\_den)



**Encapsulation**

1. **Need for encapsulation**
2. **Private Attributes**
3. **Getter and Setter Methods**
4. **Class Diagram**

class Atm:

# constructor

#special/magic/dunder methods

# instance variable

def \_\_init\_\_(self):

self.\_\_pin=" "

self.\_\_balance=0

self.\_\_menu()

def \_\_menu(self): #method can be hide by using double underscore(\_\_)

user\_input=input("""

Hello, how would you like to proceed?

1. Enter 1 to create pin.

2. Enter 2 to deposit

3. Enter 3 to withdraw

4. Enter 4 to check balance

5. Enter 5 to exit

""")

if user\_input=="1":

self.create\_pin()

elif user\_input=="2":

self.deposit()

elif user\_input=="3":

self.withdraw()

elif user\_input=="4":

self.check\_balance()

else:

print("bye")

def create\_pin(self):

self.\_\_pin = input("Enter your pin")

print("Pin is set successfully")

def deposit(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

self.\_\_balance = self.\_\_balance + amount

print("Deposit successful")

else:

print("Invalid Pin")

def withdraw(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

if amount < self.\_\_balance:

self.\_\_balance = self.balance - amount

print("Operation successfully")

else:

print("insufficient funds")

else:

print("Invalid pin")

def check\_balance(self):

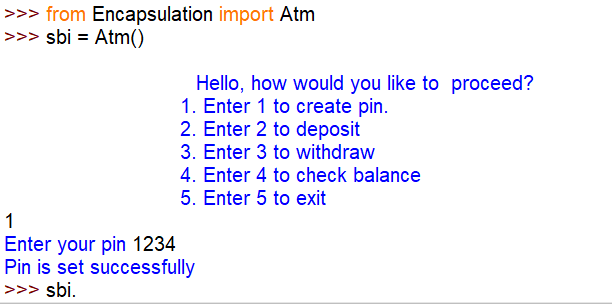
temp = input("Enter your pin")

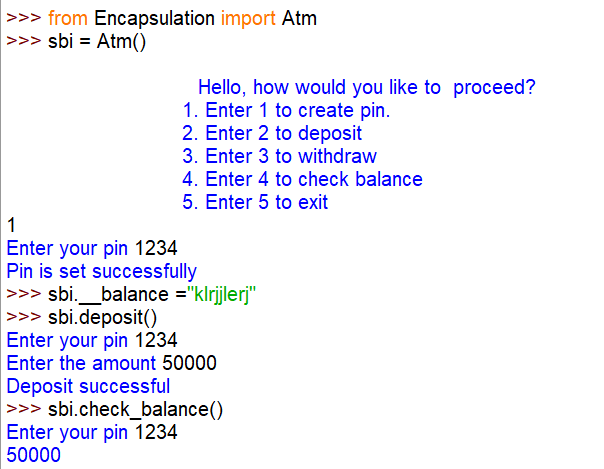
if temp == self.\_\_pin:

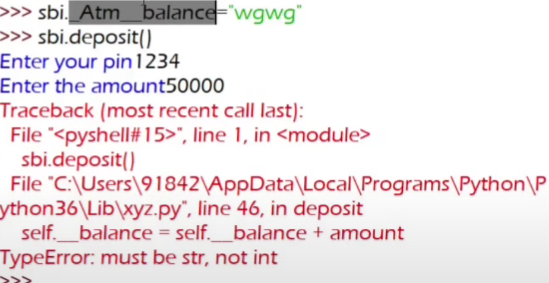
print(self.\_\_balance)

else:

print("invalid Pin")







**# Nothing in python is truly private**

**Get and set in python**

class Atm:

# constructor

#special/magic/dunder methods

# instance variable

def \_\_init\_\_(self):

self.\_\_pin=" "

self.\_\_balance=0

self.\_\_menu()

def get\_pin(self):

return self.\_\_pin

def set\_pin(self,new\_pin):

self.\_\_pin = new\_pin

print("Pin Changed")

def \_\_menu(self): #method can be hide by using double underscore(\_\_)

user\_input=input("""

Hello, how would you like to proceed?

1. Enter 1 to create pin.

2. Enter 2 to deposit

3. Enter 3 to withdraw

4. Enter 4 to check balance

5. Enter 5 to exit

""")

if user\_input=="1":

self.create\_pin()

elif user\_input=="2":

self.deposit()

elif user\_input=="3":

self.withdraw()

elif user\_input=="4":

self.check\_balance()

else:

print("bye")

def create\_pin(self):

self.\_\_pin = input("Enter your pin")

print("Pin is set successfully")

def deposit(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

self.\_\_balance = self.\_\_balance + amount

print("Deposit successful")

else:

print("Invalid Pin")

def withdraw(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

if amount < self.\_\_balance:

self.\_\_balance = self.balance - amount

print("Operation successfully")

else:

print("insufficient funds")

else:

print("Invalid pin")

def check\_balance(self):

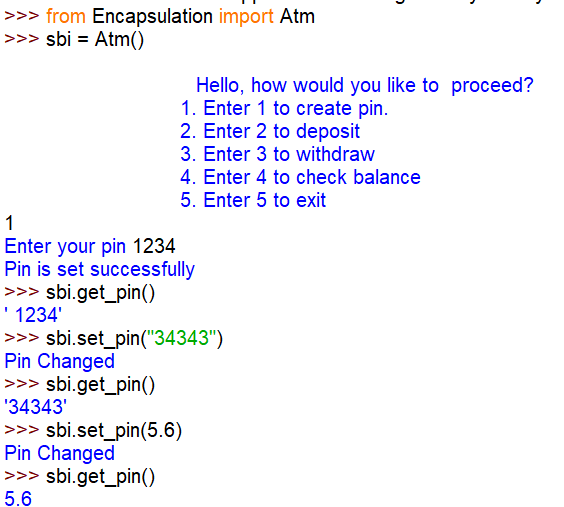
temp = input("Enter your pin")

if temp == self.\_\_pin:

print(self.\_\_balance)

else:

print("invalid Pin")



class Atm:

# constructor

#special/magic/dunder methods

# instance variable

def \_\_init\_\_(self):

self.\_\_pin=" "

self.\_\_balance=0

self.\_\_menu()

def get\_pin(self):

return self.\_\_pin

def set\_pin(self,new\_pin):

if type(new\_pin) == str:

self.\_\_pin = new\_pin

print("Pin Changed")

else:

print("Not allowed")

def \_\_menu(self): #method can be hide by using double underscore(\_\_)

user\_input=input("""

Hello, how would you like to proceed?

1. Enter 1 to create pin.

2. Enter 2 to deposit

3. Enter 3 to withdraw

4. Enter 4 to check balance

5. Enter 5 to exit

""")

if user\_input=="1":

self.create\_pin()

elif user\_input=="2":

self.deposit()

elif user\_input=="3":

self.withdraw()

elif user\_input=="4":

self.check\_balance()

else:

print("bye")

def create\_pin(self):

self.\_\_pin = input("Enter your pin")

print("Pin is set successfully")

def deposit(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

self.\_\_balance = self.\_\_balance + amount

print("Deposit successful")

else:

print("Invalid Pin")

def withdraw(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

if amount < self.\_\_balance:

self.\_\_balance = self.balance - amount

print("Operation successfully")

else:

print("insufficient funds")

else:

print("Invalid pin")

def check\_balance(self):

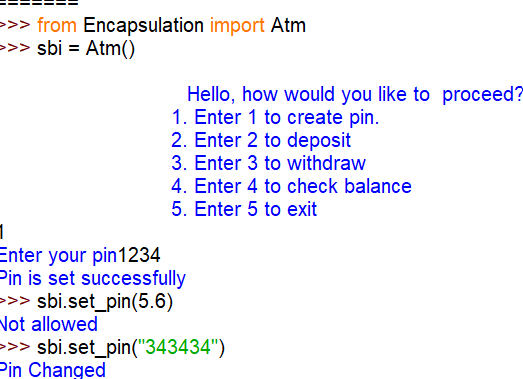
temp = input("Enter your pin")

if temp == self.\_\_pin:

print(self.\_\_balance)

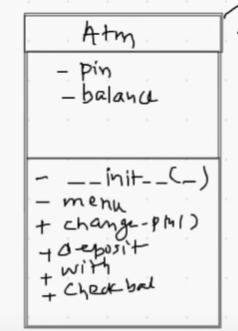
else:

print("invalid Pin")



(-) means private

(+) means public



**Pass by reference**

class Customer:

def \_\_init\_\_(self,name):

self.name = name

cust= Customer("Ram")

print(cust.name)



class Customer:

def \_\_init\_\_(self,name,gender):

self.name = name

self.gender=gender

def greet(customer):

if customer.gender=="Male":

print("Hello",customer.name,"sir")

else:

print("Hello",customer.name,"mam")

cust= Customer("Ram","Male")

greet(cust)



class Customer:

def \_\_init\_\_(self,name,gender):

self.name = name

self.gender=gender

def greet(customer):

if customer.gender=="Male":

print("Hello",customer.name,"sir")

else:

print("Hello",customer.name,"mam")

cust= Customer("Sita","Female")

greet(cust)



class Customer:

def \_\_init\_\_(self,name,gender):

self.name = name

self.gender=gender

def greet(customer):

if customer.gender=="Male":

print("Hello",customer.name,"sir")

else:

print("Hello",customer.name,"mam")

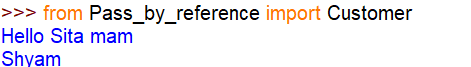
cust2 = Customer("Shyam","Male")

return cust2

cust= Customer("Sita","Female")

new\_cust = greet(cust)

print(new\_cust.name)



class Customer:

def \_\_init\_\_(self,name):

self.name = name

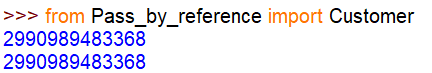
def greet(customer):

print(id(customer))

cust = Customer("Ankita")

print(id(cust))

greet(cust)



class Customer:

def \_\_init\_\_(self,name):

self.name = name

def greet(customer):

print(id(customer))

customer.name="Abc"

print(customer.name)

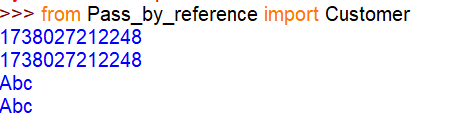
cust = Customer("Ankita")

print(id(cust))

greet(cust)

print(cust.name)

# class objects are also mutable like list, dict and sets



def change(L):

print(id(L))

L.append(5)

print(id(L))

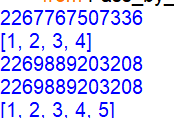
L1 = [1,2,3,4]

print(id(L1))

print(L1)

change(L1[:])

print(L1)



**Collection of Object**

class Customer:

def \_\_init\_\_(self,name,age):

self.name= name

self.age =age

def intro(self):

print("I am",self.name,"and I am",self.age)

c1 = Customer("abc",34)

c2 = Customer("xyz",55)

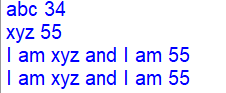
L = [c1,c2]

for i in L:

print(i.name, i.age)

for j in L:

i.intro()



**Static**

**Static Counter**

class Atm:

# constructor

#special/magic/dunder methods

# instance variable

def \_\_init\_\_(self):

self.\_\_pin=" "

self.\_\_balance=0

self.sno = 0

self.sno +=1

print(id(self))

self.\_\_menu()

def get\_pin(self):

return self.\_\_pin

def set\_pin(self,new\_pin):

self.\_\_pin = new\_pin

print("Pin Changed")

def \_\_menu(self): #method can be hide by using double underscore (\_\_)

user\_input=input("""

Hello, how would you like to proceed?

1. Enter 1 to create pin.

2. Enter 2 to deposit

3. Enter 3 to withdraw

4. Enter 4 to check balance

5. Enter 5 to exit

""")

if user\_input=="1":

self.create\_pin()

elif user\_input=="2":

self.deposit()

elif user\_input=="3":

self.withdraw()

elif user\_input=="4":

self.check\_balance()

else:

print("bye")

def create\_pin(self):

self.\_\_pin = input("Enter your pin")

print("Pin is set successfully")

def deposit(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

self.\_\_balance = self.\_\_balance + amount

print("Deposit successful")

else:

print("Invalid Pin")

def withdraw(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

if amount < self.\_\_balance:

self.\_\_balance = self.balance - amount

print("Operation successfully")

else:

print("insufficient funds")

else:

print("Invalid pin")

def check\_balance(self):

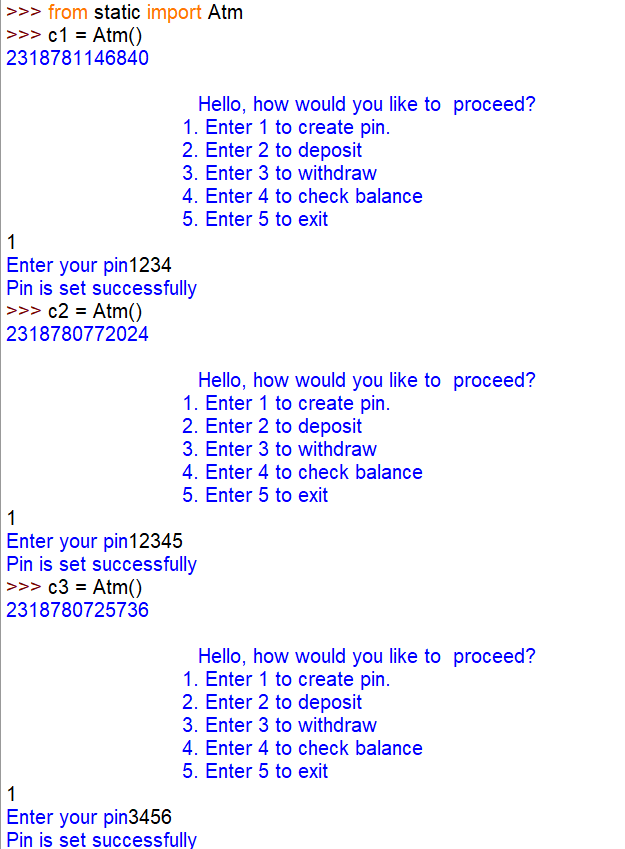
temp = input("Enter your pin")

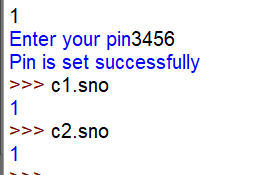
if temp == self.\_\_pin:

print(self.\_\_balance)

else:

print("invalid Pin")





class Atm:

# constructor

#special/magic/dunder methods

# instance variable

#static /class

counter =1

def \_\_init\_\_(self):

self.\_\_pin=" "

self.\_\_balance=0

self.sno = 0

self.sno +=Atm.counter

Atm.counter = Atm.counter + 1

print(id(self))

# self.\_\_menu()

def get\_pin(self):

return self.\_\_pin

def set\_pin(self,new\_pin):

self.\_\_pin = new\_pin

print("Pin Changed")

def \_\_menu(self): #method can be hide by using double underscore(\_\_)

user\_input=input("""

Hello, how would you like to proceed?

1. Enter 1 to create pin.

2. Enter 2 to deposit

3. Enter 3 to withdraw

4. Enter 4 to check balance

5. Enter 5 to exit

""")

if user\_input=="1":

self.create\_pin()

elif user\_input=="2":

self.deposit()

elif user\_input=="3":

self.withdraw()

elif user\_input=="4":

self.check\_balance()

else:

print("bye")

def create\_pin(self):

self.\_\_pin = input("Enter your pin")

print("Pin is set successfully")

def deposit(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

self.\_\_balance = self.\_\_balance + amount

print("Deposit successful")

else:

print("Invalid Pin")

def withdraw(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

if amount < self.\_\_balance:

self.\_\_balance = self.balance - amount

print("Operation successfully")

else:

print("insufficient funds")

else:

print("Invalid pin")

def check\_balance(self):

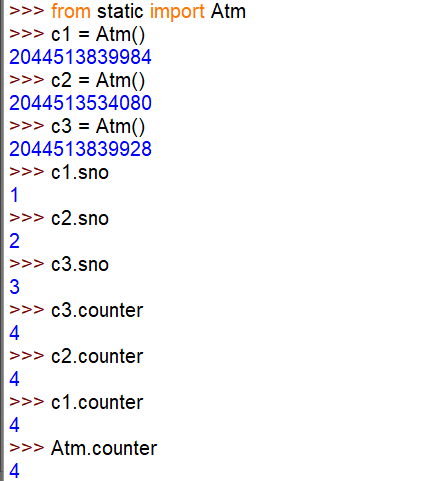
temp = input("Enter your pin")

if temp == self.\_\_pin:

print(self.\_\_balance)

else:

print("invalid Pin")



class Atm:

# constructor

#special/magic/dunder methods

# instance variable

#static /class

\_\_counter =1

def \_\_init\_\_(self):

self.\_\_pin=" "

self.\_\_balance=0

self.sno = 0

self.sno +=Atm.counter

Atm.\_\_counter = Atm.\_\_counter + 1

print(id(self))

# self.\_\_menu()

# static method representation

#static method does not require self

@staticmethod

def get\_counter():

return Atm.\_\_counter

@staticmethod

def set\_counter(new):

if type(new) == int:

Atm.\_\_counter = new

else:

print("Not allowed")

def get\_pin(self):

return self.\_\_pin

def set\_pin(self,new\_pin):

self.\_\_pin = new\_pin

print("Pin Changed")

def \_\_menu(self): #method can be hide by using double underscore(\_\_)

user\_input=input("""

Hello, how would you like to proceed?

1. Enter 1 to create pin.

2. Enter 2 to deposit

3. Enter 3 to withdraw

4. Enter 4 to check balance

5. Enter 5 to exit

""")

if user\_input=="1":

self.create\_pin()

elif user\_input=="2":

self.deposit()

elif user\_input=="3":

self.withdraw()

elif user\_input=="4":

self.check\_balance()

else:

print("bye")

def create\_pin(self):

self.\_\_pin = input("Enter your pin")

print("Pin is set successfully")

def deposit(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

self.\_\_balance = self.\_\_balance + amount

print("Deposit successful")

else:

print("Invalid Pin")

def withdraw(self):

temp = input("Enter your pin")

if temp == self.\_\_pin:

amount = int(input("Enter the amount"))

if amount < self.\_\_balance:

self.\_\_balance = self.balance - amount

print("Operation successfully")

else:

print("insufficient funds")

else:

print("Invalid pin")

def check\_balance(self):

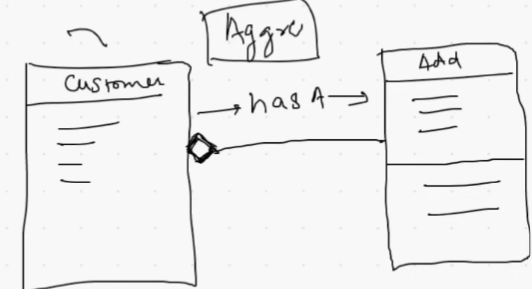
temp = input("Enter your pin")

if temp == self.\_\_pin:

print(self.\_\_balance)

else:

print("invalid Pin")



class Customer:

def \_\_init\_\_ (self,name,gender,address):

self.name = name

self.gender = gender

self.address = address

def edit\_profile(self,new\_name, new\_city,new\_pin,new\_state):

self.name = new\_name

self.address.change\_address(new\_city,new\_pin,new\_state)

class Address:

def \_\_init\_\_(self,city,pincode,state):

self.city = city

self.pincode = pincode

self.state = state

def change\_address(self,new\_city,new\_pin,new\_state):

self.city = new\_city

self.pincode = new\_pin

self.state = new\_state

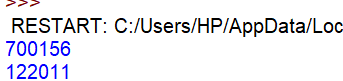
add = Address("kolkata",700156,"WB")

cust = Customer("Rohan","Male",add)

print(cust.address.pincode)

cust.edit\_profile("Ankit" ,"Guragaon",122011,"haryana")

print(cust.address.pincode)



**Inheritance**

class User:

def login(self):

print("Login")

def register(self):

print("Register")

class Student(User):

def enroll(self):

print("Enroll")

def review(self):

print("Review")

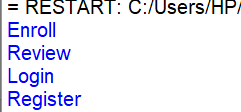
stu1 = Student()

stu1.enroll()

stu1.review()

stu1.login()

stu1.register()



class Phone:

def \_\_init\_\_(self,price,brand,camera):

print("Inside phone constructor")

self.price = price

self.brand = brand

self.camera = camera

class SmartPhone(Phone):

pass

s = SmartPhone(20000,"Apple",13)

print(s.brand)



**Inheriting Private Members**

class Phone:

def \_\_init\_\_(self,price,brand,camera):

print("Inside phone constructor")

self.price = price

self.\_\_brand = brand

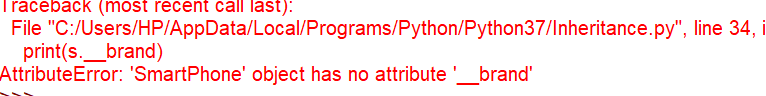
self.camera = camera

class SmartPhone(Phone):

pass

s = SmartPhone(20000,"Apple",13)

print(s.\_\_brand)



**Polymorphism**

#Method overriding -> polymorphism

class Phone:

def \_\_init\_\_(self,price,brand,camera):

print("Inside phone constructor")

self.price = price

self.brand = brand

self.camera = camera

def buy(self):

print("Buying a phone")

class SmartPhone(Phone):

def buy(self):

print("Buying a smartphone")

s = SmartPhone(20000,"Apple",13)

s.buy()



class Parent:

def \_\_init\_\_(self,num):

self.\_\_num = num

def get\_num(self):

return self.\_\_num

class Child(Parent):

def show(self):

print("This is in child class")

son = Child(100)

print(son.get\_num())

son.show()



class Parent:

def \_\_init\_\_(self,num):

self.\_\_num=num

def get\_num(self):

return self .\_\_num

class Child(Parent):

def \_\_init\_\_ (self,val,num):

self.\_\_val=val

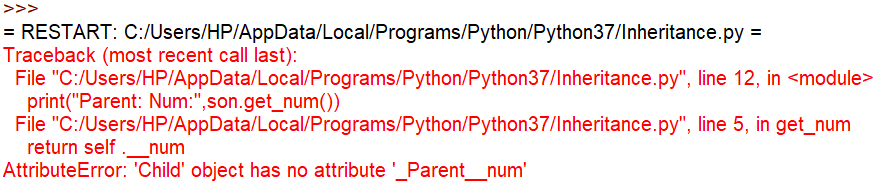
def get\_val(self):

return self .\_\_val

son=Child(100,10)

print("Parent: Num:",son.get\_num())

print("Child: Val:",son.get\_val())



class A:

def \_\_init\_\_(self):

self.var1 = 100

def display1(self,var1):

print("Class A:",self.var1)

class B(A):

def display2(self,var1):

print("class B:",self.var1)

obj = B()

obj.display1(200)



**Example of super keyword**

class Phone:

def \_\_init\_\_(self,price,brand,camera):

print("Inside phone constructor")

self.price = price

self.brand = brand

self.camera = camera

def buy(self):

print("Buying a phone")

class SmartPhone(Phone):

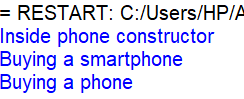
def buy(self):

print("Buying a smartphone")

super().buy()

s = SmartPhone(20000,"Apple",13)

s.buy()



class Phone:

def \_\_init\_\_(self,price,brand,camera):

print("Inside phone constructor")

self.price = price

self.brand = brand

self.camera = camera

class SmartPhone(Phone):

def \_\_init\_\_(self,price,brand,camera,os,ram):

super().\_\_init\_\_(price,brand,camera)

self.os = os

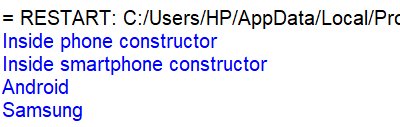
self.ram = ram

print("Inside smartphone constructor")

s = SmartPhone(20000,"Samsung",12,"Android",2)

print(s.os)

print(s.brand)



class Parent:

def \_\_init\_\_(self,num):

self.\_\_num=num

def get\_num(self):

return self.\_\_num

class Child(Parent):

def \_\_init\_\_(self,num,val):

super().\_\_init\_\_(num)

self.\_\_val=val

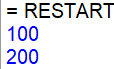
def get\_val(self):

return self .\_\_val

son=Child(100,200)

print(son.get\_num())

print(son.get\_val())



class Parent:

def \_\_init\_\_(self):

self.num=100

class Child(Parent):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.var=200

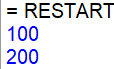
def show(self):

print(self.num)

print(self.var)

son=Child()

son.show()



class Parent:

def \_\_init\_\_(self):

self.\_\_num=100

def show(self):

print("Parent",self.\_\_num)

class Child(Parent):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.\_\_var=20

def show(self):

print("child",self.\_\_var)

dad = Parent()

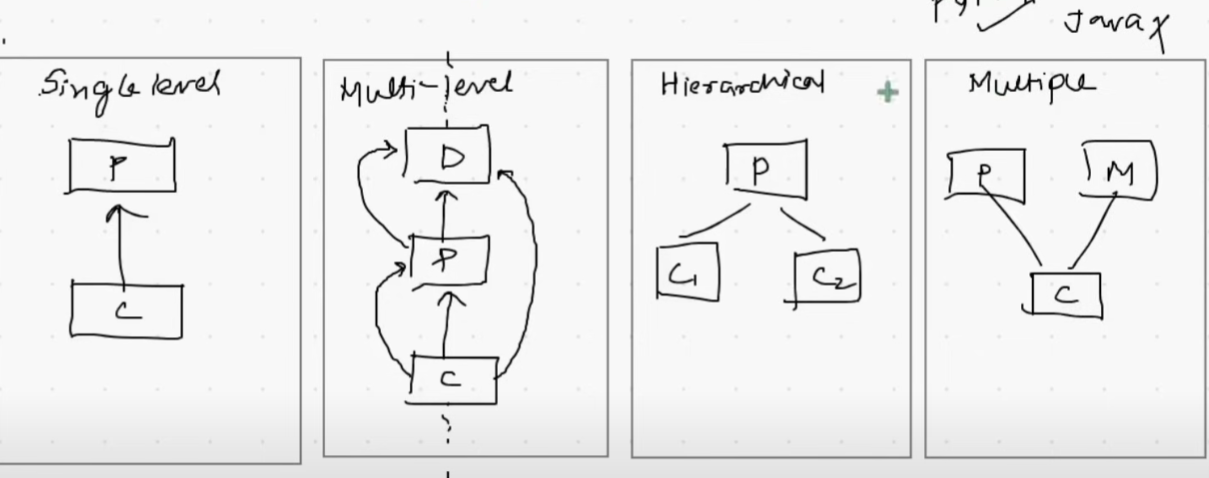
dad.show()

son=Child()

son.show()



**Types of Inheritance**



**Single Level Inheritance**

def \_\_init\_\_(self,price,brand,camera):

print("Inside phone constructor")

self.\_\_price = price

self.brand = brand

self.camera = camera

def buy(self):

print("Buying a phone")

def return\_phone(self):

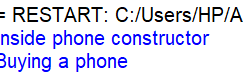
print("Returning a phone")

class SmartPhone(Phone):

pass

s = SmartPhone(20000,"Apple",13)

s.buy()



class Product:

def review(self):

print("Product customer review")

class Phone(Product):

def \_\_init\_\_(self,price,brand,camera):

print("Inside phone constructor")

self.\_\_price = price

self.brand = brand

self.camera = camera

def buy(self):

print("Buying a phone")

class SmartPhone(Phone):

pass

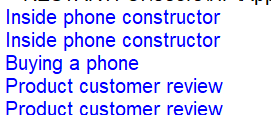
s = SmartPhone(20000,"Apple",13)

p = Phone(1000,"Samsung",1)

s.buy()

s.review()

p.review()



**Hierarchical Inheritance**

class Phone:

def \_\_init\_\_(self,price,brand,camera):

print("Inside phone constructor")

self.\_\_price = price

self.brand = brand

self.camera = camera

def buy(self):

print("Buying a phone")

def return\_phone(self):

print("Returning a Phone")

class SmartPhone(Phone):

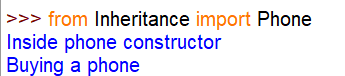
pass

class FeaturePhone(Phone):

pass

s = SmartPhone(20000,"Apple",13)

s.buy()



class Phone:

def \_\_init\_\_(self,price,brand,camera):

print("Inside phone constructor")

self.\_\_price = price

self.brand = brand

self.camera = camera

def buy(self):

print("Buying a phone")

def return\_phone(self):

print("Returning a Phone")

class Product:

def review(self):

print("Customer review")

class SmartPhone(Phone,Product):

pass

s = SmartPhone(20000,"Apple",13)

s.buy()

s.review()



class Phone:

def \_\_init\_\_(self,price,brand,camera):

print("Inside phone constructor")

self.\_\_price = price

self.brand = brand

self.camera = camera

def buy(self):

print("Buying a phone")

class Product:

def buy(self):

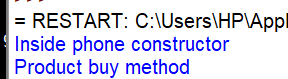
print("Product buy method")

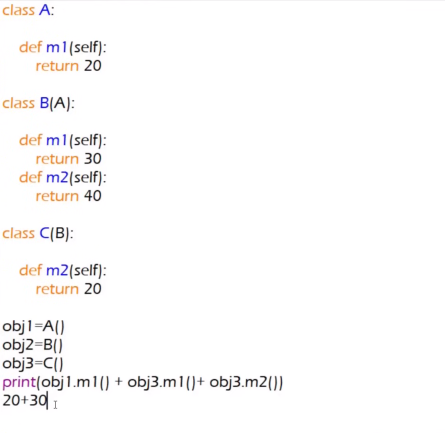
class SmartPhone(Product,Phone):

pass

s = SmartPhone(20000,"Apple",13)

s.buy()





#Method overloading in not working technical in python

class Geometry:

def area(self,radius):

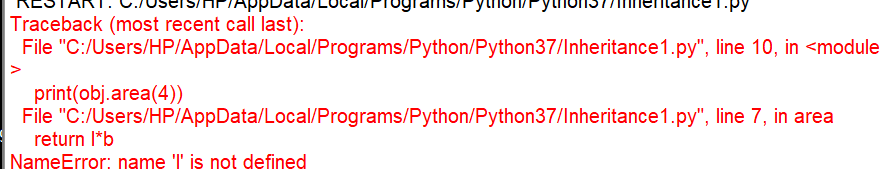
return 3.14 \* radius

def area(self,b):

return l\*b

obj = Geometry()

print(obj.area(4))



class Geometry:

def area(self,a,b=0):

if b==0:

print("Circle",3.14\*a\*a)

else:

print("Rectangle",a\*b)

obj = Geometry()

obj.area(4)

obj.area(4,5)

