In [1]:

```
class Mobile:
    pass

mob1=Mobile()
mob2=Mobile()

print(id(mob1))
print(id(mob2))
#TWO ADDRESS IS CREATED FOR THE ABOVE REFERENCE VARIABLES
```

```
2549622735624
2549622737096
```

In [ ]:

```

```

In [2]:

```
class Mobile:
    pass

mob1=Mobile()
mob2=Mobile()

mob1.price=20000
mob1.brand="Apple"

mob2.price=3000
mob2.brand="Samsung"

print (mob1.brand)
print (mob2.brand)

#ACCESSING ATTRIBUTES OF A CLASS
```

```
Apple
Samsung
```

In [3]:

```
class Mobile:
    pass

mob1=Mobile()
mob2=Mobile()

mob1.price=20000
mob1.brand="Apple"
mob1.ios_version=10

mob1.ios_version=11

mob2.price=3000
mob2.brand="Samsung"

mob2.android_version="Marshmallow"

print(mob1.ios_version)
print(mob2.android_version)
```

```
11
Marshmallow
```

In [ ]:

```
class Mobile:
    pass

mob1=Mobile()
mob2=Mobile()

mob1.price=20000
mob1.brand="Apple"
mob1.ios_version=10

mob2.price=3000
mob2.brand="Samsung"

print(mob1.ios_version)
print(mob2.ios_version)
```

In [4]:

```
class Mobile:
    def __init__(self):
        print("Inside constructor")
mob1=Mobile()
```

```
Inside constructor
```

In [5]:

```
class Mobile:
    def __init__(self):
        print("Inside constructor")


mob1=Mobile()
mob2=Mobile()
```

```
Inside constructor
Inside constructor
```

In [7]:

```
class Mobile:
    def __init__(self,one,two):
        print("Inside constructor")

#Uncomment each line below. Try it out and observe the output.

#mob1=Mobile()
mob1=Mobile(100,200,300)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-ac2d2c3148cc> in <module>
      6
      7 #mob1=Mobile()
----> 8 mob1=Mobile(100,200,300)

TypeError: __init__() takes 3 positional arguments but 4 were given
```

In [8]:

```
class Mobile:
    def __init__(self, price, brand):
        print("Id of self in constructor", id(self))
        self.price = price
        self.brand = brand

mob1=Mobile(1000, "Apple")
print("Id of mob1 in driver code", id(mob1))
#print(mob1.price,mob1.brand)

mob2=Mobile(1000, "Apple")
print("Id of mob2 in driver code", id(mob2))
#print(mob2.price,mob2.brand)
```

```
Id of self in constructor 2549622089160
Id of mob1 in driver code 2549622089160
Id of self in constructor 2549623010760
Id of mob2 in driver code 2549623010760
```

In [9]:

```
class Mobile:
    brd=90  #static variable
    def __init__(self):
        print ("Inside the Mobile constructor")
        self.brand = None
        brand = "Apple" #This is a local variable.
        #Variables without self are local and they dont
        #affect the attributes.

        #Local varaibles cannot be accessed outside the init
        #Using self creates attributes which are
        #accessible in other methods as well
        Mobile.brd+=1
    #print(brand)

mob1=Mobile()
mob2=Mobile()
mob3=Mobile()
print(mob1.brand)#This does not print Apple
#This prints None because brand=Apple creates
#a local variable and it does not affect the attribute
print(Mobile.brd)
```

```
Inside the Mobile constructor
Inside the Mobile constructor
Inside the Mobile constructor
None
93
```

In [10]:

```
class Mobile:
    def display(self):
        print("Displaying details")

    def purchase(self):
        self.display()
        print("Calculating price")

Mobile().purchase()
```

```
Displaying details
Calculating price
```

In [ ]:

```
#object created but not referenced so its useless we can't access the methods and variables

class Mobile:
    def __init__(self, price, brand):
        self.price = price
        self.brand = brand

Mobile(1000, "Apple")
#After the above line the Mobile
# object created is lost and unusable
```

In [11]:

```
#multiple references will not create new objects

class Mobile:
    def __init__(self, price, brand):
        print ("Inside constructor")
        self.price = price
        self.brand = brand

mob1=Mobile(1000, "Apple")
mob2=mob1
print ("Id of object referred by mob1 reference variable is :", id(mob1))
print ("Id of object referred by mob2 reference variable is :", id(mob2))
#mob1 and mob2 are reference variables to the same object
```

```
Inside constructor
Id of object referred by mob1 reference variable is : 2549621865032
Id of object referred by mob2 reference variable is : 2549621865032
```

In [12]:

```
class Mobile:
    def __init__(self, price, brand):
        self.price = price
        self.brand = brand

mob1=Mobile(1000, "Apple")
print("Price of mobile 1 :", mob1.price)

mob2=mob1
mob2.price=3000

print("Price of mobile 1 :", mob1.price)
print("Price of mobile 2 :", mob2.price)
```

```
Price of mobile 1 : 1000
Price of mobile 1 : 3000
Price of mobile 2 : 3000
```

In [13]:

```
class Shoe:
    def __init__(self, price, material):
        self.price = price
        self.material = material

    def __str__(self):
        return "Shoe with price: " + str(self.price) + " and material: " + self.material

print(Shoe(1000, "Canvas"))
```

```
Shoe with price: 1000 and material: Canvas
```

In [14]:

```
class Customer:
    def __init__(self, cust_id, name, age, wallet_balance):
        self.cust_id = cust_id
        self.name = name
        self.age = age
        self.wallet_balance = wallet_balance

    def update_balance(self,amount):
        if amount < 1000 and amount > 0:
            self.wallet_balance += amount

    def show_balance(self):
            print("The balance is ",self.wallet_balance)

c1=Customer(100, "Gopal", 24, 1000)
c1.update_balance(500)
c1.show_balance()
```

```
The balance is  1500
```

In [16]:

```
class Customer:
    def __init__(self, cust_id, name, age, wallet_balance):
        self.cust_id = cust_id
        self.name = name
        self.age = age
        self.__wallet_balance = wallet_balance

    def update_balance(self, amount):
        if amount < 1000 and amount > 0:
            self.__wallet_balance += amount

    def show_balance(self):
        print ("The balance is ",self.__wallet_balance)

c1=Customer(100, "Gopal", 24, 1000)
print(c1.__wallet_balance)
#print(Customer._wallet_balance)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-16-23beb7314eae> in <module>
     14
     15 c1=Customer(100, "Gopal", 24, 1000)
---> 16 print(c1.__wallet_balance)
     17 #print(Customer._wallet_balance)

AttributeError: 'Customer' object has no attribute '__wallet_balance'
```

guess the output

In [17]:

```
class Customer:
    def __init__(self, cust_id, name, age, wallet_balance):
        self.cust_id = cust_id
        self.name = name
        self.age = age
        self.__wallet_balance = wallet_balance

    def update_balance(self, amount):
        if amount < 1000 and amount > 0:
            self.__wallet_balance += amount

    def show_balance(self):
        print ("The balance is ",self.__wallet_balance)

c1=Customer(100, "Gopal", 24, 1000)
c1.__wallet_balance = 10000000000
c1.show_balance()
```

```
The balance is  1000
```

In [18]:

```
class Customer:
    def __init__(self, cust_id, name, age,wallet_balance):
        self.cust_id = cust_id
        self.name = name
        self.age = age
        self.__wallet_balance = wallet_balance

    def update_balance(self, amount):
        if amount < 1000 and amount> 0:
            self.__wallet_balance += amount

    def show_balance(self):
        print ("The balance is ",self.__wallet_balance)

c1=Customer(100, "Gopal", 24, 1000)
c1._Customer__wallet_balance = 10000000000
c1.show_balance()
```

```
The balance is  10000000000
```

In [19]:

```
class Mobile:
    discount = 50
    def __init__(self, price, brand):
        self.price = price
        self.brand = brand

    def purchase(self):
        total = self.price - self.price * Mobile.discount / 100
        print (self.brand, "mobile with price", self.price, "is available after discount at

mob1=Mobile(20000, "Apple")
mob2=Mobile(30000, "Apple")
mob3=Mobile(5000, "Samsung")

mob1.purchase()
mob2.purchase()
mob3.purchase()
```

```
Apple mobile with price 20000 is available after discount at 10000.0
Apple mobile with price 30000 is available after discount at 15000.0
Samsung mobile with price 5000 is available after discount at 2500.0
```

static methods

In [20]:

```
class Mobile:
    __discount = 50
    def __init__(self, price, brand):
        self.price = price
        self.brand = brand

    def purchase(self):
        total = self.price - self.price * Mobile.__discount / 100
        print ("Total is ",total)

    #@staticmethod
    def get_discount():
        return Mobile.__discount

    #@staticmethod
    def set_discount(discount):
        Mobile.__discount = discount

print (Mobile.get_discount())
```

```
50
```

In [21]:

```
class mobile:
    def ur(self,b):
        print("apple boi pratik",b)
a=mobile()
mobile().ur("Apple")
```

```
apple boi pratik Apple
```

In [ ]:

```
def sort(a):
    for i in range(len(l)-1):
        for j in range(i+1,len(l)):
            if a[i]>a[j]:
                a[i],a[j]=a[j],a[i]

l=[3,2,1,0,10,9,-1]
sort(l)
print(l)
```

In [ ]:

```
def f(x,y):
    x+=1
    y+=1
    print(id(x),id(y))
    print(x,y)
    print(id(a),id(b))
    print(a,b)
a=8
b=8
f(a,b)
#here x==y and a==b so python points to same mem location instead of creating new
```

In [ ]:

```
a=[1,2,3]
b=a
a[0]=90
print(b)
print(a)
```

In [ ]:

```
a=[1,2,3]
b=list(a)  #creates a new address
a[0]=0
print(a)
print(b)
```

In [ ]:

```

```

single inheritance

In [31]:

```
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):
    def __init__(self,p,s,c):
        print("inside smartphone")
        Phone(p,s,c).return_phone()

SmartPhone(1000,"Apple","13px").buy()
```

```
inside smartphone
Inside phone constructor
Returning a phone
Buying a phone
```

multilevel inheritance

In [ ]:

```
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")

    def return_phone(self):
        print ("Returning a phone")

class SmartPhone(Phone):
    pass

s=SmartPhone(20000, "Apple", 12)

s.buy()
s.review()
```

In [ ]:

```
class CreditCard:
    def __init__(self, card_no, balance):
        self.card_no = card_no
        self.balance = balance
class Customer:
    def __init__(self,cards):
        self.cards=cards
    def purchase_item(self, price, card_no):
        if price < 0:
            raise Exception()
        if card_no not in self.cards:
            raise Exception()
        if price>self.cards[card_no].balance:
            raise Exception()
card1=CreditCard(101,800)
card2=CreditCard(102,2000)
cards={card1.card_no:card1,card2.card_no:card2}
c=Customer(cards)
while(True):
    card_no=int(input("Please enter a card number"))
    try:
        c.purchase_item(-1,card_no)
        break
    except Exception as e:
        print("Something went wrong. "+str(e))
```

In [ ]:

```
class InvalidPrice(Exception):
    pass
class WrongCard(Exception):
    pass
class CreditCard:
    def __init__(self, card_no, balance):
        self.card_no=card_no
        self.balance=balance
class Customer:
    def __init__(self,cards):
        self.cards=cards
    def purchase_item(self,price,card_no):
        if price < 0:
            raise InvalidPrice("The price is wrong")
        if card_no not in self.cards:
            raise WrongCard("Card is invalid")
        if price>self.cards[card_no].balance:
            raise WrongCard("Card has insufficient balance")
card1=CreditCard(101,800)
card2=CreditCard(102,2000)
cards={card1.card_no:card1,card2.card_no:card2}
c=Customer(cards)
while(True):
    card_no=int(input("Please enter a card number"))
    try:
        c.purchase_item(500000,card_no)
        break
    except InvalidPrice as e:
        print(str(e))
        break
    except WrongCard as e:
        print(str(e))
        continue
    except Exception as e:
        print("Something went wrong. "+str(e))
```

private attribute

In [ ]:

```
class a:
    def __init__(self,b):
        self.__b=b
z=a(45)
a.__b=8 # this creates new vairable
print(z._a__b)  #object._classname__privateattributename
```

In [ ]:

```
class test:
    @staticmethod
    def sample():
        print("hello world")
class test2(test):
    @staticmethod
    def sample():
        print("infosys")
test2().sample()
a=test2
a.sample()
test2.sample()

#both works for static method
```

In [ ]:

```
class test:
    @staticmethod
    def sample():
        print("hello world")
class test2(test):

    def sample(self):
        print("infosys")
test2().sample()
print(test2)
print(test2())

#test2.sample() #error
```

In [ ]:

```

```