VISHAL ACHARYA

Write a class called Product. The class should have fields called name, amount, and price, holding the product's name, the number of items of that product in stock, and the regular price of the product. There should be a method get_price that receives the number of items to be bought and returns a the cost of buying that many items, where the regular price is charged for orders of less than 10 items, a 10% discount is applied for orders of between 10 and 99 items, and a 20% discount is applied for orders of 100 or more items. There should also be a method called make_purchase that receives the number of items to be bought and decreases amount by that much.

In [7]:

```python
class Product:

    def __init__(self, name, amount, price):
        self.name = name
        self.amount = amount
        self.price = price

    def get_price(self, number_to_be_bought):
        discount = 0
        if number_to_be_bought < 10:
            pass
        elif 10 <= number_to_be_bought < 99:
            discount = 10
        else:
            discount = 20
        price = (100 - discount) / 100 * self.price
        return price * number_to_be_bought

    def make_purchase(self, quantity):
        self.amount -= quantity


# name = input('name:')
# amount = int(input('Digit amount of items'))
# price = int(input('Digit price of items'))

name, amount, price = 'shoes', 200, 33

shoes = Product(name, amount, price)
# quantity = int(input('Digit amount of items to buy'))

q1 = 4
print(f'cost for {q1} {shoes.name} = {shoes.get_price(q1)}')
shoes.make_purchase(q1)
print(f'remaining stock: {shoes.amount}\n')

q2 = 12
print(f'cost for {q2} {shoes.name} = {shoes.get_price(q2)}')
shoes.make_purchase(q2)
print(f'remaining stock: {shoes.amount}\n')

q3 = 112
print(f'cost for {q3} {shoes.name} = {shoes.get_price(q3)}')
shoes.make_purchase(q3)
print(f'remaining stock: {shoes.amount}\n')
```

```
cost for 4 shoes = 132.0
remaining stock: 196

cost for 12 shoes = 356.4
remaining stock: 184

cost for 112 shoes = 2956.8
remaining stock: 72
```


You need to create the foundations of an e-commerce engine for a B2C (business-to-consumer) retailer. You need to have a class for a customer called User, a class for items in inventory called Item, and a shopping cart class called Cart.

Items go in Carts, and Users can have multiple Carts. Also, multiple items can go into Carts, including more than one of any single item

In [8]:

```python
class User:
    def __init__(self, id, name):
        self.id = id
        self.name = name
    def display_user(self):
        print('ID', self.id, 'Name:', self.name)

class Item:
    def __init__(self, id, name, price, sold, available):
        self.id = id
        self.name = name
        self.price = price
        self.sold = sold
        self.available = available

class Cart:
    def __init__(self, user):
        self.user = user
        self.cart_items = []
    def insert_items(self, item, quantity):
        for i in range(quantity):
            if item.available == 0:
                print('Out of stock')
                break
            self.cart_items.append(item)
            item.sold += 1
            item.available -= 1
    def display_cart(self):
        print('This Cart belongs to', self.user.name, 'with ID', self.user.id)
        self.total = 0
        for i in self.cart_items:
            print('Item',i.name)
            self.total += i.price
        print('Total price = ', self.total)



#Creating a user
user1 = User(1, 'VISHAL')
user1.display_user()

#Creating items
apple = Item(1, 'apple', 100, 0, 10)
greencoconut = Item(2, 'green coconut', 150, 0, 3)
milk = Item(2, 'milk', 24, 0, 100)

#Creating two carts for user1
cart1 = Cart(user1)
cart2 = Cart(user1)

#Adding items to cart1
cart1.insert_items(apple, 2)
cart1.insert_items(milk, 3)

#Adding items to cart2
cart2.insert_items(greencoconut, 3)
cart2.insert_items(milk, 20)

#Displaying details of cart1 and cart2
cart1.display_cart()
cart2.display_cart()
```

```
62
```

```
ID 1 Name: VISHAL
This Cart belongs to VISHAL with ID 1
Item apple
Item apple
Item milk
Item milk
Item milk
Total price =  272
This Cart belongs to VISHAL with ID 1
Item green coconut
Item green coconut
Item green coconut
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Item milk
Total price =  930
```


Get index in the list of objects by attribute in Python

In [9]:

```
class X:
    def __init__(self,val):
        self.val = val

def getIndex(li,target):
    for index, x in enumerate(li):
        if x.val == target:
            return index
    return -1

# Driver code
li = [1,2,3,4,5,6]

# Converting all the items in
# list to object of class X
a = list()
for i in li:
    a.append(X(i))

print(getIndex(a,3))
```

```
2
```


How to create a list of object in Python class

In [10]:

```
class geeks:
    def __init__(self, name, roll):
        self.name = name
        self.roll = roll

# creating list
list = []

# appending instances to list
list.append(geeks('Akash', 2))
list.append(geeks('Deependra', 40))
list.append(geeks('Reaper', 44))
list.append(geeks('veer', 67))

# Accessing object value using a for loop
for obj in list:
    print(obj.name, obj.roll, sep=' ')

print("")
# Accessing individual elements
print(list[0].name)
print(list[1].name)
print(list[2].name)
print(list[3].name)
```

```
Akash 2
Deependra 40
Reaper 44
veer 67

Akash
Deependra
Reaper
veer
```

Create a Sphere class that accepts a radius upon instantiation and has a volume and surface area method.

In [11]:
```
class Sphere():

    def __init__(self,radius):
        self.radius = radius

    def volume(self):
        return (4/3)*3.14*self.radius**3

    def surface_area(self):
        return 4 *3.14 * self.radius **2
```

In [12]:
```
s = Sphere(1)
print(s.surface_area())
print(s.volume())
```

```
12.56
4.1866666666666665
```

Create a Python class called BankAccount which represents a bank account, having as attributes: accountNumber (numeric type), name (name of the account owner as string type), balance.

Create a constructor with parameters: accountNumber, name, balance.

Create a Deposit() method which manages the deposit actions.

Create a Withdrawal() method which manages withdrawals actions.

Create an bankFees() method to apply the bank fees with a percentage of 5% of the balance account.

Create a display() method to display account details.

Give the complete code for the BankAccount class.

In [13]:

```python
class BankAccount:
    # create the constuctor with parameters: accountNumber, name and balance
    def __init__(self,accountNumber, name, balance):
        self.accountNumber = accountNumber
        self.name = name
        self.balance = balance

    # create Deposit() method
    def Deposit(self , d ):
        self.balance = self.balance + d

    # create Withdrawal method
    def Withdrawal(self , w):
        if(self.balance < w):
            print("impossible operation! Insufficient balance !")
        else:
            self.balance = self.balance - w
    # create bankFees() method
    def bankFees(self):
        self.balance = (95/100)*self.balance

    # create display() method
    def display(self):
        print("Account Number : " , self.accountNumber)
        print("Account Name : " , self.name)
        print("Account Balance : " , self.balance , " $")

# Testing the code :
newAccount = BankAccount(2178514584, "Albert" , 2700)
# Creating Withdrawal Test
newAccount.Withdrawal(300)
# Create deposit test
newAccount.Deposit(200)
# Display account informations
newAccount.display()
```

```
Account Number :  2178514584
Account Name :  Albert
Account Balance :  2600  $
```


1 - Create a Coputation class with a default constructor (without parameters) allowing to perform various calculations on integers numbers.

2 - Create a method called Factorial() which allows to calculate the factorial of an integer. Test the method by instantiating the class.

3 - Create a method called Sum() allowing to calculate the sum of the first n integers 1 + 2 + 3 + .. + n. Test this method.

4 - Create a method called testPrim() in the Calculation class to test the primality of a given integer. Test this method.

4 - Create a method called testPrims() allowing to test if two numbers are prime between them.

5 - Create a tableMult() method which creates and displays the multiplication table of a given integer. Then create an allTablesMult() method to display all the integer multiplication tables 1, 2, 3, ..., 9.

6 - Create a static listDiv() method that gets all the divisors of a given integer on new list called Ldiv. Create another listDivPrim() method that gets all the prime divisors of a given integer.

VISHAL ACHARYA

In [14]:

```
class Computation:
    def __init__ (self):
        pass
# --- Factorial ------------
    def factorial(self, n):
        j = 1
        for i in range (1, n + 1):
            j = j * i
        return j

# --- Sum of the first n numbers ----
    def sum (self, n):
        j = 1
        for i in range (1, n + 1):
            j = j + i
        return j

# --- Primality test of a number ------------
    def testPrim (self, n):
        j = 0
        for i in range (1, n + 1):
            if (n% i == 0):
                j = j + 1
        if (j == 2):
            return True
        else:
            return False

# --- Primality test of two integers ------------
    def testprims (self, n, m):

        # initialize the number of commons divisors
        commonDiv = 0
        for i in range (1, n + 1):
            if (n% i == 0 and m% i == 0):
                commonDiv = commonDiv + 1
        if commonDiv == 1:
            print ("The numbers", n, "and", m, "are co-primes")
        else:
            print ("The numbers", n, "and", m, "are not co-primes")

#---Multiplication table-------------
    def tableMult (self, k):
        for i in range (1,10):
            print (i, "x", k, "=", i * k)

# --- All multiplication tables of the numbers 1, 2, .., 9
    def allTables (self):
        for k in range (1,10):
            print ("\nthe multiplication table of:", k, "is:")
            for i in range (1,10):
                print (i, "x", k, "=", i * k)

# ----- list of divisors of an integer
    def listDiv (self, n):
        # initialization of the list of divisors
        lDiv = []
        for i in range (1, n + 1):
            if (n% i == 0):
                lDiv.append (i)
        return lDiv

# ------ list of prime divisors of an integer ----------------
```

```
    def listDivPrim (self, n):
        # initialization of the list of divisors
        lDiv = []
        for i in range (1, n + 1):
            if (n% i == 0 and self.testPrim (i)):
                lDiv.append (i)
        return lDiv

# Instantiation example
Comput= Computation ()
Comput.testprims (13, 7)
print ("List of divisors of 18:", Comput.listDiv (18))
print ("List of prime divisors of 18:", Comput.listDivPrim (18))
Comput.allTables ()
```

```
The numbers 13 and 7 are co-primes
List of divisors of 18: [1, 2, 3, 6, 9, 18]
List of prime divisors of 18: [2, 3]

the multiplication table of: 1 is:
1 x 1 = 1
2 x 1 = 2
3 x 1 = 3
4 x 1 = 4
5 x 1 = 5
6 x 1 = 6
7 x 1 = 7
8 x 1 = 8
9 x 1 = 9

the multiplication table of: 2 is:
1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18

the multiplication table of: 3 is:
1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 = 12
5 x 3 = 15
6 x 3 = 18
7 x 3 = 21
8 x 3 = 24
9 x 3 = 27

the multiplication table of: 4 is:
1 x 4 = 4
2 x 4 = 8
3 x 4 = 12
4 x 4 = 16
5 x 4 = 20
6 x 4 = 24
7 x 4 = 28
8 x 4 = 32
9 x 4 = 36

the multiplication table of: 5 is:
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
4 x 5 = 20
5 x 5 = 25
6 x 5 = 30
7 x 5 = 35
8 x 5 = 40
9 x 5 = 45

the multiplication table of: 6 is:
1 x 6 = 6
2 x 6 = 12
3 x 6 = 18
```

```
4 x 6 = 24
5 x 6 = 30
6 x 6 = 36
7 x 6 = 42
8 x 6 = 48
9 x 6 = 54

the multiplication table of: 7 is:
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63

the multiplication table of: 8 is:
1 x 8 = 8
2 x 8 = 16
3 x 8 = 24
4 x 8 = 32
5 x 8 = 40
6 x 8 = 48
7 x 8 = 56
8 x 8 = 64
9 x 8 = 72

the multiplication table of: 9 is:
1 x 9 = 9
2 x 9 = 18
3 x 9 = 27
4 x 9 = 36
5 x 9 = 45
6 x 9 = 54
7 x 9 = 63
8 x 9 = 72
9 x 9 = 81
```


A university wants to automate their admission process. Students are admitted based on the marks scored in the qualifying exam. A student is identified by student id, age and marks in qualifying exam. Data are valid, if:

Age is greater than 20

Marks is between 0 and 100 (both inclusive)

A student qualifies for admission, if

Age and marks are valid and

Marks is 65 or more

Write a python program to represent the students seeking admission in the university. The details of student class are given below.

Class name: Student

Attributes (private)	student_id marks age	
Methods (public)	__init__()	Create and initialize all instance variables to None
	validate_marks()	If data is valid, return true. Else, return false
	validate_age()	

	check_qualification()	Validate marks and age.
		• If valid, check if marks is 65 or more.
		  ◦ If so return true
		  ◦ Else return false
		• Else return false

	setter methods	Include setter methods for all instance variables to set its values
	getter methods	Include getter methods for all instance variables to get its values

In [15]:

```
class Student:

  def __init__(self):
    self.__sid = None
    self.__marks = None
    self.__age = None

  # setter methods
  def set_sid(self,sid):
    self.__sid = sid
  def set_marks(self,marks):
    self.__marks = marks
  def set_age(self,age):
    self.__age = age

  # getters
  def get_sid(self):
    return self.__sid
  def get_marks(self):
    return self.__marks
  def get_age(self):
    return self.__age

  def validate_age(self):
    return self.__age>20

  def validate_marks(self):
    return self.__marks>=0 and self.__marks<=100

  def check_qualification(self):
    if self.validate_age() and self.validate_marks():
      return self.__marks>= 65
    else:
      return False

stu1 = Student()

stu1.set_sid(101)
stu1.set_marks(66)
stu1.set_age(19)

print(stu1.get_sid())
print(stu1.get_marks())
print(stu1.get_age())

stu1.check_qualification()
```

```
101
66
19
```

Out[15]:

```
False
```


Ice-Cream Scoops and Bowl shop

1. Create a class `Scoop` which has one public property `flavor` and one private proptery `price`. Take `flavor` values during object creation.
2. Create a class `Bowl` with private prperty `scoop_list` which will have list of scoopd object.
3. Create a method `add_scoops` in `Bowl` class which will add any no of `Scoop` objects given as parameter and store it in `scoops_list`.

4. Make getter and setter method for `price` property.
5. Make a method `display` to display `flavour` and `price` of each `Scoop` in `scoop_list` and print total price of the bowl by adding all flavour scoops prices.
6. Make a method `sold` in both `Scoop` class and `Bowl` class to print no of quantity sold.

In [16]:

```python
#Write your code here

class Scoop:

  __counter = 0

  def __init__(self,flavor):
    self.flavor = flavor
    self.__price = None
    Scoop.__counter += 1


  def get_price(self):
    return self.__price

  def set_price(self,price):
    self.__price = price

  def __str__(self):
    return "Flavor - {} and Price - {}".format(self.flavor,self.__price)

  @staticmethod
  def sold():
    return Scoop.__counter


class Bowl:

  __counter = 0

  def __init__(self):
    self.__scoop_list = []
    Bowl.__counter += 1

  def add_scoops(self,*new_scoops):
    for scoop in new_scoops:
      self.__scoop_list.append(scoop)

  def display(self):
    total = 0
    for scoop in self.__scoop_list:
      print(scoop)
      total = total + scoop.get_price()

    print('total price',total)

  @staticmethod
  def sold():
    return Bowl.__counter

```

In [17]:

```
choco = Scoop('chocolate')
print(choco)
choco.set_price(100)

berry = Scoop('berry')
berry.set_price(120)
print(berry)

vanilla = Scoop('vanilla')
vanilla.set_price(150)

bowl = Bowl()

bowl.add_scoops(choco) # Giving one parameter
bowl.add_scoops(berry, vanilla) # Multiple
# add_scoops should handle both scenerios

#print(bowl)

bowl.display()

Scoop.sold()
Bowl.sold()
```

```
Flavor - chocolate and Price - None
Flavor - berry and Price - 120
Flavor - chocolate and Price - 100
Flavor - berry and Price - 120
Flavor - vanilla and Price - 150
total price 370
```

Out[17]: 1

Ice-Cream Bowl continue..

Making advancement in the above classes. `Scoop` and `Bowl`

1. Introduce a property `max_scoops` in `Bowl` class to signify maximum scoops that a bowl can have, exceeding that it will display `Bowl is full` . Take default value as `3` .
2. `no_of_scoop` in `Scoop` class with default value of `1`
3. Print `<flavour> added` with every scoop added.

In [26]:

```
#Write your code here

class Scoop:

  __counter = 0

  def __init__(self,flavor,num_scoops=1):
    self.flavor = flavor
    self.__price = None
    self.num_scoops = num_scoops
    Scoop.__counter += 1


  def get_price(self):
    return self.__price

  def set_price(self,price):
    self.__price = price

  def __str__(self):
    return "Flavor - {} and Price - {}".format(self.flavor,self.__price)

  @staticmethod
  def sold():
    return Scoop.__counter


class Bowl:

  __counter = 0

  def __init__(self,max_scoops=3):
    self.__scoop_list = []
    Bowl.__counter += 1
    self.scoops_added = 0
    self.max_scoops = max_scoops

  def add_scoops(self,*new_scoops):
    for scoop in new_scoops:
      if self.scoops_added + scoop.num_scoops <= self.max_scoops:
        self.__scoop_list.append(scoop)
        self.scoops_added = self.scoops_added + scoop.num_scoops
        print(scoop.flavor,'added!')
      else:
        print('Bowl is full')
        break

  def display(self):
    total = 0
    for scoop in self.__scoop_list:
      print(scoop)
      total = total + scoop.get_price()

    print('total price',total)

  @staticmethod
  def sold():
    return Bowl.__counter

```

In [19]:

```
choco = Scoop('chocolate', 1)
choco.set_price(100)
print(choco)


berry = Scoop('berry', 2)
berry.set_price(120)
print(berry)

vanilla = Scoop('vanilla') # no of scoop parameter not given, will take default v
vanilla.set_price(150)
print(vanilla)

bowl1 = Bowl() # max_scoop parameter not given, will take default value
bowl1.add_scoops(choco) # Giving one parameter
bowl1.add_scoops(berry, vanilla) # Multiple
bowl1.display()
```

```
Flavor - chocolate and Price - 100
Flavor - berry and Price - 120
Flavor - vanilla and Price - 150
chocolate added!
berry added!
Bowl is full
Flavor - chocolate and Price - 100
Flavor - berry and Price - 120
total price 220
```

In [20]:

```
#write a python program that has class store which keeps record of code and price
#each product. Display a menu of all products to the user and prompt
#him to enter the quantity of each item required . generate a bill and display to
class Store:
    def __init__(self,n):
        self.item_code=[]
        self.price=[]
        self.n=n
    def get_data(self):
        #n=int(input("enter no of items: "))
        for i in range(self.n):
            self.item_code.append(int(input("enter code of item: ")))
            self.price.append(int(input("enter cost of item: ")))
    def display_data(self):
        print("Item Code \t Price")
        for i in range(self.n):
            print(self.item_code[i],"\t\t ",self.price[i])

    def calculate_bill(self,quant):
        total_amount=0
        for i in range(self.n):
            total_amount+=(self.price[i]*quant[i])
        print("**********************Bill**********************")
        print("ITEM \t PRICE \t QUANTITY \t SUBTOTAL")
        for i in range(self.n):
            print(self.item_code[i],"\t",self.price[i],"\t",quant[i],"\t\t",
                  self.price[i]*quant[i])
        print("************************************")
        print("Total= ",total_amount)

n=int(input("enter no of items: "))
s1=Store(n)
s1.get_data()
s1.display_data()
q=[]
print("Enter quantity of each item: ")
for i in range(n):
    q.append(int(input("Enter quantity of item {} : ".format(i+1))))

s1.calculate_bill(q)

```

```
enter no of items: 1
enter code of item: 1
enter cost of item: 40
Item Code        Price
1                 40
Enter quantity of each item:
Enter quantity of item 1 : 4
**********************Bill**********************
ITEM     PRICE    QUANTITY        SUBTOTAL
1        40       4               160
************************************
Total=  160
```

In [21]:

```python
#write a python program for library book record with oops.

class library:
    def __init__(self):
        self.title=""
        self.author=""
        self.publisher=""
    def read(self):
        self.title=input("Enter Book Title: ")
        self.author=input("Enter Book author: ")
        self.publisher=input("Enter Book Publisher: ")
    def display(self):
        print("Title:", self.title)
        print("Author:", self.author)
        print("Publisher:", self.publisher)
        print("\n")
my_book=[]
ch='y'
while(ch=='y'):
    print('''
1. Add New Book
2. Display Books
''')
    choice=int(input("Enter choice: "))
    if(choice==1):
        book=library()
        book.read()
        my_book.append(book)
    elif(choice==2):
        for i in my_book:
            i.display()
    else:
        print("Invalid choice!")
    ch=input("Do you want to continue..?")
print("Bye!")
```



```
1. Add New Book
2. Display Books

Enter choice: 1
Enter Book Title: H
Enter Book author: L
Enter Book Publisher: O
Do you want to continue..?NO
Bye!
```

Write a class called Investment with fields called principal and interest. The constructor should set the values of those fields. There should be a method called value_after that returns the value of the investment after n years. The formula for this is p(1+i)n, where p is the principal, and i is the interest rate. It should also use the special method str so that printing the object will result in something like below:

- Principal - $1000.00, Interest rate - 5.12%

In [23]:

```python
class BankAccount:
    def __init__(self, name, amount, interest_rate):
        self.name = name
        self.amount = amount
        self.interest_rate = interest_rate

    def apply_interest(self):
        self.amount *= (1 + self.interest_rate / 100)

account = BankAccount('VISHAL', 10000, 3)
account.apply_interest()
print(account.amount)
account.interest_rate = 2
account.apply_interest()
print(account.amount)
```

```
10300.0
10506.0
```