Q. 584

Write a program to build a simple Student Management System using Object Oriented Programming in Python which can perform the following operations: accept-This method takes details from the user like name, roll number, and marks for two different subjects. display-This method displays the details of every student. search-This method searches for a particular student from the list of students. This method will ask the user for roll number and then search according to the roll number delete-This method deletes the record of a particular student with a matching roll number. update-This method updates the roll number of the student. This method will ask for the old roll number and new roll number. It will replace the old roll number with a new roll number. The following instructions need to be considered while making a program.

1. Give class name as Student
2. Include methods name as accept, display, search, delete and update. (1 mark for each correct method to be formed).
3. Also form constructor with **init** () method (2 marks for forming constructor).
4. 2 marks for correct object prepared like after deletion of one roll no of student it should update the list with new roll no. and should display it. The example is just for understanding but logic should be for any n number of students. For Example: List of Students Name : A RollNo : 1 Marks1 : 100 Marks2 : 100 Name : B RollNo : 2 Marks1 : 90 Marks2 : 90 Name : C RollNo : 3 Marks1 : 80 Marks2 : 80 Student Found, Name : B RollNo : 2 Marks1 : 90 Marks2 : 90 List after deletion Name : A RollNo : 1 Marks1 : 100 Marks2 : 100 Name : C RollNo : 3 Marks1 : 80 Marks2 : 80 List after updation Name : A RollNo : 1 Marks1 : 100 Marks2 : 100 Name : C RollNo : 2 Marks1 : 80

In [1]:
```
# This is simplest Student data management program in python

# Create class "Student"
class Student:

# Constructor
        def __init__(self, name, rollno, m1, m2):
                self.name = name
                self.rollno = rollno
                self.m1 = m1
                self.m2 = m2

        # Function to create and append new student
        def accept(self, Name, Rollno, marks1, marks2):

# use ' int(input()) ' method to take input from user
                ob = Student(Name, Rollno, marks1, marks2)
                ls.append(ob)

        # Function to display student details
        def display(self, ob):
                print("Name : ", ob.name)
                print("RollNo : ", ob.rollno)
                print("Marks1 : ", ob.m1)
                print("Marks2 : ", ob.m2)
                print("\n")
```

```
        # Search Function
        def search(self, rn):
                for i in range(ls.__len__()):
                        if(ls[i].rollno == rn):
                                return i

        # Delete Function
        def delete(self, rn):
                i = obj.search(rn)
                del ls[i]

        # Update Function
        def update(self, rn, No):
                i = obj.search(rn)
                roll = No
                ls[i].rollno = roll


# Create a list to add Students
ls = []
# an object of Student class
obj = Student('', 0, 0, 0)

print("\nOperations used, ")
print("\n1.Accept Student details\n2.Display Student Details\n3.Search Details of a

# ch = int(input("Enter choice:"))
# if(ch == 1):
obj.accept("A", 1, 100, 100)
obj.accept("B", 2, 90, 90)
obj.accept("C", 3, 80, 80)

# elif(ch == 2):
print("\n")
print("\nList of Students\n")
for i in range(ls.__len__()):
        obj.display(ls[i])

# elif(ch == 3):
print("\n Student Found, ")
s = obj.search(2)
obj.display(ls[s])

# elif(ch == 4):
obj.delete(2)
print(ls.__len__())
print("List after deletion")
for i in range(ls.__len__()):
        obj.display(ls[i])

# elif(ch == 5):
obj.update(3, 2)
print(ls.__len__())
print("List after updation")
for i in range(ls.__len__()):
        obj.display(ls[i])

# else:
print("Thank You !")
```

Operations used,

1.Accept Student details
2.Display Student Details
3.Search Details of a Student
4.Delete Details of Student
5.Update Student Details
6.Exit

List of Students

Name : A
RollNo : 1
Marks1 : 100
Marks2 : 100

Name : B
RollNo : 2
Marks1 : 90
Marks2 : 90

Name : C
RollNo : 3
Marks1 : 80
Marks2 : 80

Student Found,
Name : B
RollNo : 2
Marks1 : 90
Marks2 : 90

2
List after deletion
Name : A
RollNo : 1
Marks1 : 100
Marks2 : 100

Name : C
RollNo : 3
Marks1 : 80
Marks2 : 80

2
List after updation
Name : A
RollNo : 1
Marks1 : 100
Marks2 : 100

Name : C
RollNo : 2
Marks1 : 80

```
Marks2 :  80

Thank You !
```

Q. 585 You own a pizzeria named Olly's Pizzas and want to create a Python program to handle the customers and revenue. Create

the following classes with the following methods: Class Pizza containing

1. init method: to initialize the size (small, medium, large), toppings (corn, tomato, onion, capsicum, mushroom, olives, broccoli), cheese (mozzarella, feta, cheddar). Note: One pizza can have only one size but many toppings and cheese. (1.5 marks) Throw custom exceptions if the selects toppings or cheese not available in lists given above. (1 mark)
2. price method: to calculate the prize of the pizza in the following way: small = 50, medium = 100, large = 200 Each topping costs 20 rupees extra, except broccoli, olives and mushroom, which are exotic and so cost 50 rupees each. Each type of cheese costs an extra 50 rupees. (1.5 marks) Class Order containing
3. init method: to initialize the name, customerid of the customer who placed the order (0.5 marks)
4. order method: to allow the customer to select pizzas with choice of toppings and cheese (1 mark)
5. bill method: to generate details about each pizza ordered by the customer and the total cost of the order. (2 marks) *Note: A customer can get multiple pizzas in one order. 1.5 marks for creating appropriate objects of these classes and writing correct output.

In [2]:
```
class Pizza:

  def __init__(self, size, toppings, cheese):
    self.size = size
    self.toppings = toppings
    self.cheese = cheese

  def price(self):
    self.cost = 0
    if self.size == 'small':
      self.cost += 50
    elif self.size == 'medium':
      self.cost += 100
    else:
      self.cost += 200
    topping_prices_20 = ['corn', 'tomato', 'onion', 'capsicum']
    topping_prices_50 = ['mushroom', 'olives', 'broccoli']
    for topping in self.toppings:
      if topping in topping_prices_20:
        self.cost += 20
      else:
        self.cost += 50
        #for cheese
    self.cost += 50 * len(self.cheese)
    return self.cost
```

```
class Order:
  def __init__(self, name, customerid):
    self.name = name
    self.customerid = customerid

  def order(self, n):
    """n is the number of pizzas to order"""
    self.pizzas = []
    for i in range(n):
      toppings = []
      cheese = []
      print('Customize Pizza',i+1)
      size = input('Select size: small, medium or large: ')
      t = int(input('How many toppings: '))
      #take input for toppings
      print("Available Toppings: corn, tomato, onion, capsicum, mushroom, olives, b
      for i in range(t):
        toppings.append(input('Enter toppings: '))
      print("Available Types of Cheese: cheddar, mozzarella, parmigiano, feta")
      t = int(input('How many types of cheese do you want?: '))
      for i in range(t):
        cheese.append(input('Enter cheese: '))
      self.pizzas.append(Pizza(size, toppings, cheese))

  def bill(self):
    self.total = 0
    count = 1
    for p in self.pizzas:
      print('Pizza', count)
      print(p.size, p.toppings, p.cheese)
      self.total += p.price()
      count += 1
    print('Total bill amount:', self.total)

order1 = Order('Jyovita', 1)
order1.order(2)
order1.bill()
```

```
Customize Pizza 1
Select size: small, medium or large: small
How many toppings: 3
Available Toppings: corn, tomato, onion, capsicum, mushroom, olives, broccoli
Enter toppings: corn
Enter toppings: tomato
Enter toppings: onion
Available Types of Cheese: cheddar, mozzarella, parmigiano, feta
How many types of cheese do you want?: 1
Enter cheese: cheddar
Customize Pizza 2
Select size: small, medium or large: medium
How many toppings: 4
Available Toppings: corn, tomato, onion, capsicum, mushroom, olives, broccoli
Enter toppings: mushroom
Enter toppings: olives
Enter toppings: broccoli
Enter toppings: corn
Available Types of Cheese: cheddar, mozzarella, parmigiano, feta
How many types of cheese do you want?: 3
Enter cheese: feta
Enter cheese: cheddar
Enter cheese: parmigiano
Pizza 1
small ['corn', 'tomato', 'onion'] ['cheddar']
Pizza 2
medium ['mushroom', 'olives', 'broccoli', 'corn'] ['feta', 'cheddar', 'parmigian
o']
Total bill amount: 580
```

Q. 586 Write a class called WordPlay. It should have a constructor that holds a list of words. The user of the class should pass the

list of words through constructor, which user wants to use for the class. The class should have following methods: words_with_length(length) — returns a list of all the words of length length starts_with(char1) — returns a list of all the words that start with char1 ends_with(char2) — returns a list of all the words that end with char2 palindromes() — returns a list of all the palindromes in the list only(str1) — returns a list of the words that contain only those letters in str1 avoids(str2) — returns a list of the words that contain none of the letters in str2 Make Required object for WordPlay class and test all the methods. For Example: If input list entered by user is: ['apple', 'banana', 'find', 'dictionary', 'set', 'tuple', 'list', 'malayalam', 'nayan', 'grind', 'apricot'] words_with_length (5) should return ['apple', 'tuple', 'nayan', 'grind'] starts_with ('a') should return ['apple', 'apricot'] ends_with ('d') should return ['find', 'grind'] palindromes () should return ['malayalam', 'nayan'] only ('bna') should return ['banana'] avoids ('amkd') should return ['set', 'tuple', 'list']

In [3]:
```
class WordPlay:
    def __init__(self, words):
        self.words = words

    def words_with_length(self, length):
        return list(filter(lambda word: len(word) == length, self.words))
```

```
    def starts_with(self, char1):
        return list(filter(lambda word: word.startswith(char1), self.words))

    def ends_with(self, char2):
        return list(filter(lambda word: word.endswith(char2), self.words))

    def palindromes(self):
        return list(filter(lambda word: word == word[::-1], self.words))

    def only(self, str1):
        return list(filter(lambda word: set(str1).issuperset(set(word)), self.words

    def avoids(self, str2):
        return list(filter(lambda word: not any(char in word for char in str2), sel

# Example usage:
word_list = ['apple', 'banana', 'find', 'dictionary', 'set', 'tuple', 'list', 'mala
word_play = WordPlay(word_list)

print(word_play.words_with_length(5))
print(word_play.starts_with('a'))
print(word_play.ends_with('d'))
print(word_play.palindromes())
print(word_play.only('bna'))
print(word_play.avoids('amkd'))
```

```
['apple', 'tuple', 'nayan', 'grind']
['apple', 'apricot']
['find', 'grind']
['malayalam', 'nayan']
['banana']
['set', 'tuple', 'list']
```

Q. 587. Write a python program that has class store which keeps record of code and price of 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 total amount. Sample Output: enter no of items: 3 enter code of item: milk enter cost of item: 30 enter code of item: apple enter cost of item: 35 enter code of item: gems enter cost of item: 40 Item Code Price milk 30 apple 35 gems 40 Enter quantity of each item: Enter quantity of milk : 2 Enter quantity of apple : 3 Enter quantity of gems : 4 **Bill** ITEM PRICE QUANTITY SUBTOTAL milk 30 2 60 apple 35 3 105 gems 40 4 160

Total= 325

In [5]:
```
class Store:
    def __init__(self):
        self.products = {}

    def add_product(self, code, price):
        self.products[code] = price

    def display_menu(self):
        print("Item Code\t Price")
        for code, price in self.products.items():
            print(f"{code}\t\t {price}")
```

```
    def generate_bill(self, quantities):
        print("\n**********************Bill*********************")
        print("ITEM\t PRICE QUANTITY\t SUBTOTAL")
        total = 0
        for code, quantity in quantities.items():
            price = self.products[code]
            subtotal = price * quantity
            total += subtotal
            print(f"{code}\t {price}\t {quantity}\t {subtotal}")

        print("************************************")
        print(f"Total= {total}")


# Example usage:
store = Store()

# Adding products to the store
store.add_product("milk", 30)
store.add_product("apple", 35)
store.add_product("gems", 40)

# Displaying the menu
store.display_menu()

# Taking user input for quantities
num_items = int(input("\nEnter number of items: "))
quantities = {}
for _ in range(num_items):
    code = input("Enter code of item: ")
    quantity = int(input(f"Enter quantity of {code}: "))
    quantities[code] = quantity

# Generating and displaying the bill
store.generate_bill(quantities)
```

```
Item Code        Price
milk             30
apple            35
gems             40

Enter number of items: 3
Enter code of item: milk
Enter quantity of milk: 2
Enter code of item: apple
Enter quantity of apple: 3
Enter code of item: gems
Enter quantity of gems: 4

**********************Bill*********************
ITEM     PRICE QUANTITY  SUBTOTAL
milk     30      2       60
apple    35      3       105
gems     40      4       160
************************************
Total= 325
```

Q. 588. Write a python program that has a class Point with attributes as the x and y co-ordinates.

1. Add a method ‘distance from origin’ to class Point which returns the distance of the given point from origin. The equation is
2. Add a method ‘translate’ to class Point, which returns a new position of point after translation
3. Add a method ‘reflect_x’ to class Point, which returns a new point which is the reflection of the point about the x-axis.
4. Add a method ‘distance’ to return distance of the given point with respect to the other point. The formula for calculating distance between A(x1,y1) and B(x2,y2) is After creating class blueprint run the following test case - Test Case – Point (1,2) Distance from origin - 2.23 Translate method - point (1,2) translated by (1,1) increment will be at (2,3) now Reflect_x Method - Point (2,3) after given reflection will be at (2,-3) Distance Method - distance between point (2,-3) and (3,4) is 1.41

In [12]:
```
import math

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance_from_origin(self):
        return math.sqrt(self.x ** 2 + self.y ** 2)

    def translate(self, dx, dy):
        return Point(self.x + dx, self.y + dy)

    def reflect_x(self):
        return Point(self.x, -self.y)

    def distance(self, other_point):
        return math.sqrt((self.x - other_point.x) ** 2 + (self.y - other_point.y) *

# Test Case
point = Point(1, 2)

# Testing distance_from_origin method
distance_from_origin = point.distance_from_origin()
print(f"Distance from origin: {distance_from_origin:.2f}")

# Testing translate method
translation_point = point.translate(1, 1)
print(f"Translate method - Point ({point.x},{point.y}) translated by (1,1) will be

# Testing reflect_x method after translation
reflection_after_translation = translation_point.reflect_x()
print(f"Reflect_x Method - Point ({translation_point.x},{translation_point.y}) afte

# Testing distance method after reflection
other_point = Point(3, 4)
distance_between_points = reflection_after_translation.distance(other_point)
print(f"Distance Method - distance between point ({reflection_after_translation.x},
```

```
Distance from origin: 2.24
Translate method - Point (1,2) translated by (1,1) will be at (2,3) now
Reflect_x Method - Point (2,3) after given reflection will be at (2,-3)
Distance Method - distance between point (2,-3) and (3,4) is 7.07
```

Q. 589 A possible collection of classes which can be used to represent a music collection (for example, inside a music player),

focusing on how they would be related by composition. You should include classes for songs, artists, albums and playlists. For simplicity you can assume that any song or album has a single “artist” value (which could represent more than one person), but you should include compilation albums (which contain songs by a selection of different artists). The “artist” of a compilation album can be a special value like “Various Artists”. You can also assume that each song is associated with a single album, but that multiple copies of the same song (which are included in different albums) can exist. Write a simple implementation of this model which clearly shows how the different classes are composed. Write some example code to show how you would use your classes to create an album and add all its songs to a playlist. Class Album should have a method to add track, class Artist should have methods to add album and add song, class Playlist should also have a method to add song.

In [13]:
```
class Song:
    def __init__(self, title, duration):
        self.title = title
        self.duration = duration

class Artist:
    def __init__(self, name):
        self.name = name
        self.albums = []

    def add_album(self, album):
        self.albums.append(album)

    def add_song(self, song, album):
        album.add_track(song)

class Album:
    def __init__(self, title, artist):
        self.title = title
        self.artist = artist
        self.tracks = []

    def add_track(self, song):
        self.tracks.append(song)

class Playlist:
    def __init__(self, name):
        self.name = name
        self.songs = []

    def add_song(self, song):
        self.songs.append(song)

# Example usage
artist1 = Artist("Artist 1")
artist2 = Artist("Various Artists")
```

```
album1 = Album("Album 1", artist1)
album2 = Album("Compilation Album", artist2)

song1 = Song("Song 1", 180)
song2 = Song("Song 2", 200)
song3 = Song("Song 3", 220)

artist1.add_album(album1)
artist2.add_album(album2)

album1.add_track(song1)
album1.add_track(song2)

album2.add_track(song1)
album2.add_track(song3)

playlist = Playlist("My Playlist")
playlist.add_song(song1)
playlist.add_song(song2)
playlist.add_song(song3)

# Displaying information
print(f"Artist 1's albums: {[album.title for album in artist1.albums]}")
print(f"Various Artists' albums: {[album.title for album in artist2.albums]}")

print(f"\nSongs in Album 1: {[track.title for track in album1.tracks]}")
print(f"Songs in Compilation Album: {[track.title for track in album2.tracks]}")

print(f"\nSongs in My Playlist: {[song.title for song in playlist.songs]}")
```

```
Artist 1's albums: ['Album 1']
Various Artists' albums: ['Compilation Album']

Songs in Album 1: ['Song 1', 'Song 2']
Songs in Compilation Album: ['Song 1', 'Song 3']

Songs in My Playlist: ['Song 1', 'Song 2', 'Song 3']
```

590 Stacks and Queues. Write a class SQ that defines a data structure that can behave as both a queue (FIFO) or a stack (LIFO),

There are five methods that should be implemented:

1. make a constructor with a valid parameter
2. shift() returns the first element and removes it from the list. Also, use the custom(raise) exception in this method.
3. unshift() "pushes" a new element to the front or head of the list
4. push() adds a new element to the end of a list
5. pop() returns the last element and removes it from the list
6. remove() returns the maximum element of the list and removes it from the list.
7. Create the object and call all methods of the SQ class.

In [14]:
```
class EmptyListError:
    def __init__(self, message="The list is empty"):
```

```
        self.message = message

class SQ:
    def __init__(self, data):
        self.data = data

    def shift(self):
        if not self.data:
            raise EmptyListError()
        return self.data.pop(0)

    def unshift(self, element):
        self.data.insert(0, element)

    def push(self, element):
        self.data.append(element)

    def pop(self):
        if not self.data:
            raise EmptyListError()
        return self.data.pop()

    def remove(self):
        if not self.data:
            raise EmptyListError()
        max_element = max(self.data)
        self.data.remove(max_element)
        return max_element

# Example usage
try:
    sq = SQ([1, 2, 3, 4, 5])

    print("Original List:", sq.data)

    print("Shifted Element:", sq.shift())
    print("List after Shift:", sq.data)

    sq.unshift(10)
    print("List after Unshift:", sq.data)

    sq.push(20)
    print("List after Push:", sq.data)

    print("Popped Element:", sq.pop())
    print("List after Pop:", sq.data)

    print("Removed Maximum Element:", sq.remove())
    print("List after Remove:", sq.data)

except EmptyListError as e:
    print(e.message)
```

```
Original List: [1, 2, 3, 4, 5]
Shifted Element: 1
List after Shift: [2, 3, 4, 5]
List after Unshift: [10, 2, 3, 4, 5]
List after Push: [10, 2, 3, 4, 5, 20]
Popped Element: 20
List after Pop: [10, 2, 3, 4, 5]
Removed Maximum Element: 10
List after Remove: [2, 3, 4, 5]
```

In [ ]: