Unit 1

Challenge 1Q

Implement a recursive function to calculate the factorial of a given number.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Input from the user
num = int(input("Enter a number: "))

# Call the factorial function and print the result
result = factorial(num)
print(f"The factorial of {num} is {result}")
```

Output:

Enter a number:5

The factorial of 5 is 120

Challenge 2 Q

Write a program that determines whether a year entered by the user is a leap year or not using ifelif-else statements.

```
year = int(input("Enter a year: "))

# Check if it's a leap year
if year % 4 == 0:
    if year % 100 == 0:
        if year % 400 == 0:
            print(f"{year} is a leap year.")
        else:
            print(f"{year} is not a leap year.")
    else:
        print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

Output :

Certainly, here's the expected output of the program when you run it and provide a year as input:

```
Enter a year: 2024
2024 is a leap year.
```

This output indicates that the year 2024 is a leap year.

Unit 2

Challenge 1 Q

Implement a class called BankAccount that represents a bank account. The class should have private attributes for account number, account holder name, and account balance. Include methods to deposit money, withdraw money, and display the account balance. Ensure that the account balance cannot be accessed directly from outside the class. Write a program to create an instance of the BankAccount class and test the deposit and withdrawal functionality.

```
class BankAccount:
    def __init__(self, account_number, account_holder_name, initial_balance=0):
        self.__account_number = account_number
        self.__account_holder_name = account_holder_name
        self.__account_balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.__account_balance += amount
            return f"${amount} deposited successfully."
        else:
            return "Invalid deposit amount. Please enter a positive amount."

    def withdraw(self, amount):
        if amount > 0 and amount <= self.__account_balance:
            self.__account_balance -= amount
            return f"${amount} withdrawn successfully."
        else:
            return "Invalid withdrawal amount or insufficient balance."
```

```
    def display_balance(self):
        return f"Account Holder:
{self.__account_holder_name}\nAccount
Number: {self.__account_number}\nAccount
Balance: ${self.__account_balance}"

# Example usage:
if __name__ == "__main__":
    # Creating a bank account
    my_account = BankAccount("123456",
"Archana", 1000)

    # Depositing money
    print(my_account.deposit(500))  # Output:
$500 deposited successfully.

    # Withdrawing money
    print(my_account.withdraw(300))  # Output:
$300 withdrawn successfully.
    print(my_account.withdraw(1200))  #
Output: Invalid withdrawal amount or
insufficient balance.

    # Displaying the account balance
    print(my_account.display_balance())
```

Output:

$500 deposited successfully.

$300 withdrawn successfully.

Invalid withdrawal amount or insufficient balance.

Account Holder: Archana

Account Number: 123456

Account Balance: $200

Challenge 2 Q:

- Implement a class called Player that represents a cricket player. The Player class should have a method called play() which prints "The player is playing cricket. Derive two classes, Batsman and Bowler, from the Player class. Override the play() method in each derived class to print "The batsman is batting" and "The bowler is bowling", respectively. Write a program to create objects of both the Batsman and Bowler classes and call the play() method for each object.

```
class Player:
    def play(self):
        print("The player is playing cricket.")

class Batsman(Player):
    def play(self):
        print("The batsman is batting.")

class Bowler(Player):
    def play(self):
        print("The bowler is bowling.")

# Create objects of Batsman and Bowler classes and call the play() method for each object.
if __name__ == "__main__":
    batsman = Batsman()
    bowler = Bowler()

    batsman.play()  # Output: The batsman is batting.
    bowler.play()   # Output: The bowler is bowling.
```

Output:

The batsman is batting.

The bowler is bowling.

Unit 3

Challenge 1 Q

Write a function called linear_search_product that takes the list o⁷f products and a target product name as input. The function should perform a linear search to find the target product in the list and return a list of indices of all occurrences of the product if found, or an empty list if the product is not found.

```
def linear_search_product(product_list,
target_product):
    indices = []

    for index, product in
enumerate(product_list):
        if product == target_product:
            indices.append(index)

    return indices

# Example usage:
products = ["Apple", "Banana", "Apple",
"Orange", "Mango", "Apple" ]
target = "Apple"

result = linear_search_product(products,
target)
print(result)
```

Output: [0, 2, 5]

Challenge 2 Q

Implement a function called sort_students that takes a list of student objects as input and sorts the list based on their CGPA (Cumulative Grade Point Average) in descending order. Each student object has the following attributes: name (string), roll_number (string), and cgpa (float). Test the function with different input lists of students.

```
class Student:
    def __init__(self, name, roll_number, cgpa):
        self.name = name
        self.roll_number = roll_number
        self.cgpa = cgpa

def sort_students(student_list):
    sorted_students = sorted(student_list,
key=lambda student: student.cgpa,
reverse=True)
    return sorted_students

# Example usage:
if __name__ == "__main__":
    students = [
        Student("Archana", "A101", 3.8),
        Student("Akshu", "B102", 3.5),
        Student("Nithi", "C103", 4.0),
        Student("Aruna", "D104", 3.9),
    ]

    sorted_students = sort_students(students)

    for student in sorted_students:
        print(f"Name: {student.name}, Roll
Number: {student.roll_number}, CGPA:
{student.cgpa}")
```

Output

```
Name: Nithi, Roll Number: C103, CGPA: 4.0
Name: Aruna, Roll Number: D104, CGPA: 3.9
Name: Archana, Roll Number: A101, CGPA: 3.8
Name: Akshu, Roll Number: B102, CGPA: 3.5
```