**CHALLENGES**

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

def fact(n):

"""This is a recursive function

to find the factorial of an integer"""

if n == 0:

return 1

else:

return (n \* fact(n-1))

fact(5)

2. 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 year to be checked: "))

**if** year % 4 == 0:

**if** year % 100 == 0:

**if** year % 400 == 0:

**print**("The year is a leap year!")

**else**:

**print**("The year is not a leap year!")

**else**:

**print**("The year is a leap year!")

**else**:

**print**("The year is not a leap year!")

3. 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

print(f"Deposited {amount} units. New balance: {self.\_\_account\_balance}")

else:

print("Deposit amount must be greater than zero.")

def withdraw(self, amount):

if 0 < amount <= self.\_\_account\_balance:

self.\_\_account\_balance -= amount

print(f"Withdrew {amount} units. New balance: {self.\_\_account\_balance}")

else:

print("Withdrawal amount must be greater than zero and less than or equal to the account balance.")

def display\_balance(self):

print(f"Account Holder: {self.\_\_account\_holder\_name}")

print(f"Account Number: {self.\_\_account\_number}")

print(f"Account Balance: {self.\_\_account\_balance}")

# Testing the BankAccount class

if \_\_name\_\_ == "\_\_main\_\_":

account = BankAccount("1234567890", "John Doe", 1000)

account.display\_balance() # Should display the initial balance

account.deposit(500) # Depositing 500 units

account.withdraw(200) # Withdrawing 200 units

account.display\_balance() # Should display the updated balance after transactions

4. 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.")

# Testing the Player, Batsman, and Bowler classes

if \_\_name\_\_ == "\_\_main\_\_":

player = Player()

player.play() # Output: The player is playing cricket.

batsman = Batsman()

batsman.play() # Output: The batsman is batting.

bowler = Bowler()

bowler.play() # Output: The bowler is bowling.

5. 1-Write a function called linear\_search\_product that takes the list of 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.

Sure, here's a Python function called linear\_search\_product that performs a linear search to find all occurrences of a target product name in a list of products:

python

Copy code

def linear\_search\_product(products, target\_product):

indices = []

for index, product in enumerate(products):

if product == target\_product:

indices.append(index)

return indices

You can use this function by passing the list of products and the target product name as arguments. It will return a list of indices where the target product appears in the original list, or an empty list if the target product is not found.

Example usage:

python

Copy code

# Sample list of products

product\_list = ["apple", "banana", "orange", "apple", "grape", "apple"]

# Target product to search for

target\_product = "apple"

# Call the function

result = linear\_search\_product(product\_list, target\_product)

# Print the result

print(result) # Output: [0, 3, 5]

In this example, the target product "apple" appears at indices 0, 3, and 5 in the product\_list, so the function returns the list [0, 3, 5].

6. 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

Sure, here's a Python function called sort\_students that sorts a list of student objects based on their CGPA (Cumulative Grade Point Average) in descending order:

python

Copy code

def sort\_students(student\_list):

sorted\_students = sorted(student\_list, key=lambda student: student.cgpa, reverse=True)

return sorted\_students

You can use this function by passing a list of student objects as input. Each student object should have attributes: name (string), roll\_number (string), and cgpa (float). The function will sort the students based on their CGPA in descending order and return the sorted list.

Here's an example of how to use the function:

python

Copy code

class Student:

def \_\_init\_\_(self, name, roll\_number, cgpa):

self.name = name

self.roll\_number = roll\_number

self.cgpa = cgpa

# Test with different input lists of students

students\_list = [

Student("John", "2021001", 3.9),

Student("Jane", "2021002", 3.7),

Student("Alice", "2021003", 3.8),

Student("Bob", "2021004", 3.6)

]

sorted\_students = sort\_students(students\_list)

# Print the sorted list of students

for student in sorted\_students:

print(f"Name: {student.name}, Roll Number: {student.roll\_number}, CGPA: {student.cgpa}")

Output:

yaml

Copy code

Name: John, Roll Number: 2021001, CGPA: 3.9

Name: Alice, Roll Number: 2021003, CGPA: 3.8

Name: Jane, Roll Number: 2021002, CGPA: 3.7

Name: Bob, Roll Number: 2021004, CGPA: 3.6

In this example, the list of students is sorted based on their CGPA in descending order.