Example :

Name : Karthick .V

Stu Login Id : tvu23722bcs023

Pwd : 363222

NM Id : [288308F37E92CD28FC649B06A2616616@smartinternz.com](mailto:288308F37E92CD28FC649B06A2616616@smartinternz.com)

Pwd : CVndRsmE

**Challenge 1.1**

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

"""

1! = 1 x 1

2! = 2 x 1! --->2 x 1

3! = 3 x 2! --->3 x 2 x 1

10! = 10 x 9! ---> 10 x 9 x 8 x... x 1

Formula - n x (n-1)!

"""

def fact\_rec(n):

if n==0 or n==1:

return 1

else:

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

number = int(input("Enter a value : "))

res = fact\_rec(number)

print("The factorial of {} is {}.".format(number,res))

**Challenge 1.2**

# Leap year

"""

year % 4 == 0 &

year % 100 != 0 /

year % 400 == 0

"""

def isLeapYear(year):

if (year % 4 == 0 and year % 100 != 0) or year % 400 == 0:

return True

else:

return False

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

if isLeapYear(year):

print('{} is a leap year.'.format(year))

else:

print('{} is not a leap year.'.format(year))

**Challenge 2.1**

'''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=0.0):

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

# self.\_\_account\_balance = self.\_\_account\_balance+amount

print("Deposited ₹{}. New balance: ₹{}".format(amount,

self.\_\_account\_balance))

else:

print("Invalid deposit amount.")

def withdraw(self, amount):

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

self.\_\_account\_balance -= amount

# self.\_\_account\_balance = self.\_\_account\_balance - amount

print("Withdrew ₹{}. New balance: ₹{}".format(amount,

self.\_\_account\_balance))

else:

print("Invalid withdrawal amount or insufficient balance.")

def display\_balance(self):

print("Account balance for {} (Account #{}): ₹{}".format(

self.\_\_account\_holder\_name, self.\_\_account\_number,

self.\_\_account\_balance))

# Create an instance of the BankAccount class

account = BankAccount(account\_number="123456789",

account\_holder\_name="Hari Prabu",

initial\_balance=5000.0)

# Test deposit and withdrawal functionality

account.display\_balance()

account.deposit(500.0)

account.withdraw(200.0)

account.withdraw(20000.0)

account.display\_balance()

**Challenge 2.2**

'''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.'''

# Define the base class Player

class Player:

def play(self):

print("The player is playing cricket.")

# Define the derived class Batsman

class Batsman(Player):

def play(self):

print("The batsman is batting.")

# Define the derived class Bowler

class Bowler(Player):

def play(self):

print("The bowler is bowling.")

# Create objects of Batsman and Bowler classes

batsman = Batsman()

bowler = Bowler()

# Call the play() method for each object

batsman.play()

bowler.play()

**Challenge 3.1**

"""

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.

"""

def linearSearchProduct(productList, targetProduct):

indices = []

for index, product in enumerate(productList):

if product == targetProduct:

indices.append(index)

return indices

# Example usage:

products = ["shoes", "boot", "loafer", "shoes", "sandal", "shoes"]

target = "shoes"

target2 = 'apple'

result = linearSearchProduct(products, target)

print(result)

**Challenge 3.2**

'''

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):

# Sort the list of students in descending order of CGPA

sorted\_students = sorted(student\_list,

key=lambda student: student.cgpa,

reverse=True)

# Syntax - lambda arg:exp

return sorted\_students

# Example usage:

students = [

Student("Hari", "A123", 7.8),

Student("Srikanth", "A124", 8.9),

Student("Saumya", "A125", 9.1),

Student("Mahidhar", "A126", 9.9),

]

sorted\_students = sort\_students(students)

# Print the sorted list of students

for student in sorted\_students:

print("Name: {}, Roll Number: {}, CGPA: {}".format(student.name,

student.roll\_number,

student.cgpa))