1. Python Programming

1.1. Create a simple calculator in Python.

first\_no=int(input('enter the first number: '))

second\_no=int(input('enter the second number: '))

operation=input('enter the operation: supported operations are +, -, \*, /:')

if operation=='+':

    print("Addition:",first\_no + second\_no)

elif operation=='-':

    print('Substraction:',first\_no - second\_no)

elif operation=='\*':

    print('Multiplication:',first\_no \* second\_no)

elif operation=='/':

    if second\_no != 0:

        print('Division:',first\_no / second\_no)

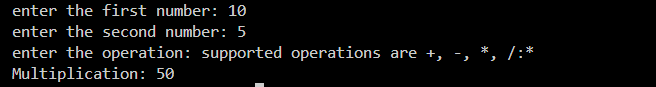
    else:

        print("Error: Division by zero is not allowed")

else:

    print("Invalid operation")

output



1.2. An electric power distribution company charges domestic customers as

follows: Consumption unit Rate of charge:

1.2.1. 0-200 Rs. 0.50 per unit

1.2.2. 201-400 Rs. 0.65 per unit in excess of 200

1.2.3. 401-600 Rs 0.80 per unit excess of 400

1.2.4. 601 and above Rs 1.00per unit excess of 600

1.2.5. If the bill exceeds Rs. 400, then a surcharge of 15% will be charged,

and the minimum bill should be Rs. 100/-

Create a Python program based on the scenario mentioned above.

def calculate\_electricity\_bill(units):

    total\_bill = 0

    if units <= 200:

        total\_bill = units \* 0.50

    elif units <= 400:

        total\_bill = 200 \* 0.50 + (units - 200) \* 0.65

    elif units <= 600:

        total\_bill = 200 \* 0.50 + 200 \* 0.65 + (units - 400) \* 0.80

    else:

        total\_bill = 200 \* 0.50 + 200 \* 0.65 + 200 \* 0.80 + (units - 600) \* 1.00

    if total\_bill > 400:

        total\_bill \*= 1.15

    if total\_bill < 100:

        total\_bill = 100

    return total\_bill

units = int(input("Enter the number of units consumed: "))

bill = calculate\_electricity\_bill(units)

print("The electricity bill is: Rs.", bill)

Output



1.3. Print the pyramid of numbers using for loops.

rows = int(input("Enter the number of rows: "))

for i in range(1, rows+1):

    for j in range(rows-i):

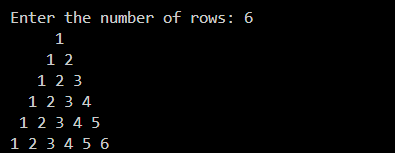
        print(" ", end="")

    for j in range(1, i+1):

        print(j, end=" ")

    print()

Output



1.4. Write a program to find the number and sum of all integers greater than 100

and less than 200 that are divisible by 7.

count = 0

total\_sum = 0

for num in range(101, 200):

    if num % 7 == 0:

        count += 1

        total\_sum += num

print("The number of integers greater than 100 and less than 200 that are divisible by 7 is:", count)

print("The sum of these integers is:", total\_sum)

Output



1.5. Write a recursive function to calculate the sum of numbers from 0 to 10

def calculate\_sum(n):

    if n == 0:

        return 0

    else:

        return n + calculate\_sum(n-1)

result = calculate\_sum(10)

print("The sum of numbers from 0 to 10 is:", result)

Output



1.6. Write a Python program to reverse the digits of a given number and add them

to the original. If the sum is not a palindrome, repeat this procedure.

def is\_palindrome(num):

"""

Checks if a number is a palindrome.

"""

original\_num = num

reversed\_num = 0

while num > 0:

digit = num % 10

reversed\_num = reversed\_num \* 10 + digit

num //= 10

return original\_num == reversed\_num

def reverse\_and\_add(num):

"""

Reverses the digits of a number and adds them to the original.

"""

original\_num = num

reversed\_num = 0

while num > 0:

digit = num % 10

reversed\_num = reversed\_num \* 10 + digit

num //= 10

return original\_num + reversed\_num

def main():

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

while not is\_palindrome(num):

num = reverse\_and\_add(num)

print("The final palindrome is:", num)

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

main()

Output



1.7. Write a menu-driven program that performs the following operations on

strings

1.7.1. Check if the String is a Substring of Another String

1.7.2. Count Occurrences of Character

1.7.3. Replace a substring with another substring

1.7.4. Convert to Capital Letters

n = 0

while n != 5:

    n = int(input("Enter your choice\n1: Check if the string is a substring of another string\n2: Count the occurrence of a character\n3: Replace a substring with another substring\n4: Convert to capital letters\n5: Exit\n"))

    if n == 1:

        # *Check if the String is a Substring of Another String*

        main\_string = input("Enter the main string: ")

        sub\_string = input("Enter the substring to check: ")

        if sub\_string in main\_string:

            print(f"'{sub\_string}' is a substring of '{main\_string}'.")

        else:

            print(f"'{sub\_string}' is not a substring of '{main\_string}'.")

    elif n == 2:

        # *Count Occurrences of Character*

        main\_string = input("Enter the main string: ")

        char = input("Enter the character to count: ")

        count = main\_string.count(char)

        print(f"Number of occurrences of '{char}' in '{main\_string}': {count}")

    elif n == 3:

        # *Replace a substring with another substring*

        main\_string = input("Enter the main string: ")

        old\_substring = input("Enter the substring to replace: ")

        new\_substring = input("Enter the new substring: ")

        modified\_string = main\_string.replace(old\_substring, new\_substring)

        print(f"Modified string: '{modified\_string}'")

    elif n == 4:

        # *Convert to Capital Letters*

        main\_string = input("Enter the string to convert to capital letters: ")

        capital\_string = main\_string.upper()

        print(f"String in capital letters: '{capital\_string}'")

    elif n == 5:

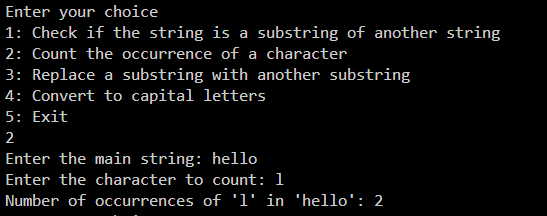
        print("Exiting the program...")

    else:

        print("Invalid choice. Please enter a number from 1 to 5.")

print("Program ended.")

Output



1.8. Write a function to find the factorial of a number but also store the factorials

calculated in a dictionary.

def factorial(n):

    """

    Calculates the factorial of a number and stores the calculated factorials in a dictionary.

    """

    factorials = {}

    def \_factorial(x):

        if x in factorials:

            return factorials[x]

        elif x == 0:

            return 1

        else:

            result = x \* \_factorial(x-1)

            factorials[x] = result

            return result

    return \_factorial(n)

print("factorial is:",factorial(5))

print("factorial is:",factorial(10))

Output



1.9. Perform various set operations

1.9.1. Set Union

1.9.2. Set Intersection

1.9.3. Set Difference

def set\_union(set1, set2):

    return set1.union(set2)

def set\_intersection(set1, set2):

    return set1.intersection(set2)

def set\_difference(set1, set2):

    return set1.difference(set2)

# *Example usage:*

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

    set1 = {1, 2, 3, 4}

    set2 = {3, 4, 5, 6}

    print("Set 1:", set1)

    print("Set 2:", set2)

    choice = 0

    while choice != 4:

        print("\nChoose an operation:")

        print("1: Set Union")

        print("2: Set Intersection")

        print("3: Set Difference")

        print("4: Exit")

        choice = int(input("Enter your choice (1-4): "))

        if choice == 1:

            print("Union of Set 1 and Set 2:", set\_union(set1, set2))

        elif choice == 2:

            print("Intersection of Set 1 and Set 2:", set\_intersection(set1, set2))

        elif choice == 3:

            print("Difference (Set 1 - Set 2):", set\_difference(set1, set2))

        elif choice == 4:

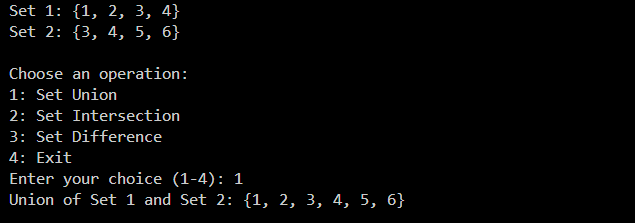
            print("Exiting the program...")

        else:

            print("Invalid choice. Please enter a number from 1 to 4.")

    print("Program ended.")

Output



1.10. Create a dictionary to store the name, roll\_no, and total\_mark of N students.

Now print the details of the student with the highest total\_mark.

def main():

    n = int(input("Enter the number of students: "))

    student\_dict = {}

    for i in range(1, n + 1):

        print(f"\nEnter details for student {i}:")

        name = input("Enter student's name: ")

        roll\_no = input("Enter student's roll number: ")

        total\_marks = float(input("Enter student's total marks: "))

        student\_dict[i] = {

            'name': name,

            'roll\_no': roll\_no,

            'total\_marks': total\_marks

        }

    highest\_marks\_student = None

    highest\_marks = -1

    for student\_id, details in student\_dict.items():

        if details['total\_marks'] > highest\_marks:

            highest\_marks = details['total\_marks']

            highest\_marks\_student = details

    if highest\_marks\_student:

        print("\nDetails of the student with the highest total marks:")

        print(f"Name: {highest\_marks\_student['name']}")

        print(f"Roll Number: {highest\_marks\_student['roll\_no']}")

        print(f"Total Marks: {highest\_marks\_student['total\_marks']}")

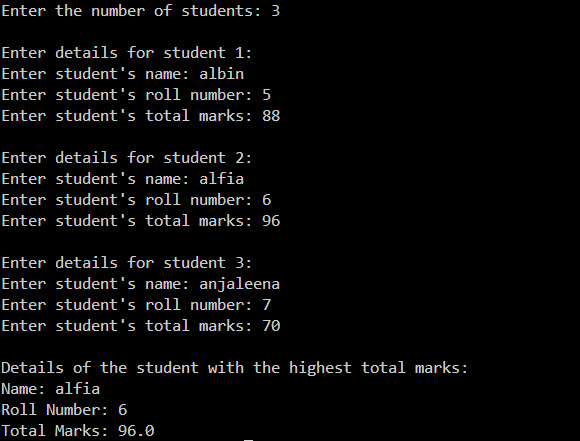
    else:

        print("\nNo student records found.")

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

    main()

Output



1.11. Write a Python program to copy the contents of a file into another file, line by

line.

def copy\_file\_contents(source\_file, destination\_file):

try:

with open(source\_file, 'r') as src:

with open(destination\_file, 'w') as dest:

for line in src:

dest.write(line)

print(f"Contents copied from {source\_file} to {destination\_file}")

except FileNotFoundError:

print(f"The file {source\_file} does not exist.")

except Exception as e:

print(f"An error occurred: {e}")

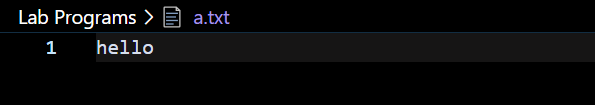
source = a.txt'

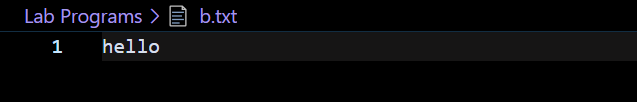
destination = b.txt'

copy\_file\_contents(source, destination)

Output







1.12. Use the OS module to perform

1.12.1. Create a directory

1.12.2. Directory Listing

1.12.3. Search for “.py” files

1.12.4. Remove a particular file

Program

import os

def create\_directory(directory\_name):

    try:

        os.makedirs(directory\_name, exist\_ok=True)

        print(f"Directory '{directory\_name}' created successfully.")

    except Exception as e:

        print(f"Error creating directory '{directory\_name}': {e}")

def list\_directory\_contents(directory\_name):

    try:

        contents = os.listdir(directory\_name)

        print(f"Contents of directory '{directory\_name}':")

        for item in contents:

            print(item)

    except FileNotFoundError:

        print(f"The directory '{directory\_name}' does not exist.")

    except Exception as e:

        print(f"Error listing contents of directory '{directory\_name}': {e}")

def search\_py\_files(directory\_name):

    try:

        py\_files = [f for f in os.listdir(directory\_name) if f.endswith('.py')]

        print(f"'.py' files in directory '{directory\_name}':")

        for file in py\_files:

            print(file)

    except FileNotFoundError:

        print(f"The directory '{directory\_name}' does not exist.")

    except Exception as e:

        print(f"Error searching for '.py' files in directory '{directory\_name}': {e}")

def remove\_file(file\_path):

    try:

        os.remove(file\_path)

        print(f"File '{file\_path}' removed successfully.")

    except FileNotFoundError:

        print(f"The file '{file\_path}' does not exist.")

    except Exception as e:

        print(f"Error removing file '{file\_path}': {e}")

directory\_name = 'Euphoria Website'

file\_name = 'a.txt'

file\_path = os.path.join(directory\_name, file\_name)

create\_directory(directory\_name)

list\_directory\_contents(directory\_name)

sample\_py\_file = os.path.join(directory\_name, 'sample.py')

with open(sample\_py\_file, 'w') as f:

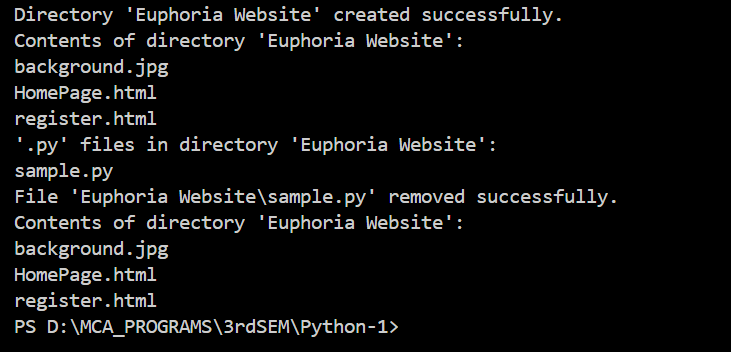
    f.write('# Sample Python file')

search\_py\_files(directory\_name)

remove\_file(sample\_py\_file)

list\_directory\_contents(directory\_name)

Output



1.13. Create a simple banking application by using inheritance.

Program

class BankAccount:

    def \_\_init\_\_(self, account\_number, account\_holder):

        self.account\_number = account\_number

        self.account\_holder = account\_holder

        self.balance = 0.0

    def deposit(self, amount):

        if amount > 0:

            self.balance += amount

            print(f"Deposited {amount}. New balance: {self.balance}")

        else:

            print("Deposit amount must be positive.")

    def withdraw(self, amount):

        if 0 < amount <= self.balance:

            self.balance -= amount

            print(f"Withdrew {amount}. New balance: {self.balance}")

        else:

            print("Invalid withdraw amount or insufficient funds.")

    def check\_balance(self):

        print(f"Account balance: {self.balance}")

class SavingsAccount(BankAccount):

    def \_\_init\_\_(self, account\_number, account\_holder, interest\_rate):

        super().\_\_init\_\_(account\_number, account\_holder)

        self.interest\_rate = interest\_rate

    def add\_interest(self):

        interest = self.balance \* self.interest\_rate / 100

        self.balance += interest

        print(f"Interest added: {interest}. New balance: {self.balance}")

class CheckingAccount(BankAccount):

    def \_\_init\_\_(self, account\_number, account\_holder, overdraft\_limit):

        super().\_\_init\_\_(account\_number, account\_holder)

        self.overdraft\_limit = overdraft\_limit

    def withdraw(self, amount):

        if 0 < amount <= self.balance + self.overdraft\_limit:

            self.balance -= amount

            print(f"Withdrew {amount}. New balance: {self.balance}")

        else:

            print("Invalid withdraw amount or exceeds overdraft limit.")

savings = SavingsAccount("A100", "Albin", 5.0)

savings.deposit(1000)

savings.check\_balance()

savings.add\_interest()

checking = CheckingAccount("A200", "Joseph", 500)

checking.deposit(500)

checking.check\_balance()

checking.withdraw(800)

checking.check\_balance()

Output

