**1. Python Programming**

**1.1. Create a simple calculator in Python.**

def add(x, y):

return x + y

def sub(x, y):

return x - y

def mul(x, y):

return x \* y

def div(x, y):

if y != 0:

return x / y

else:

return "Error: Division by zero"

print("Select operation:")

print("1. Add")

print("2. Subtract")

print("3. Multiply")

print("4. Divide")

while True:

choice = input("Enter choice (1/2/3/4): ")

if choice in ('1', '2', '3', '4'):

try:

num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))

if choice == '1':

print(f"{num1} + {num2} = {add(num1, num2)}")

elif choice == '2':

print(f"{num1} - {num2} = {sub(num1, num2)}")

elif choice == '3':

print(f"{num1} \* {num2} = {mul(num1, num2)}")

elif choice == '4':

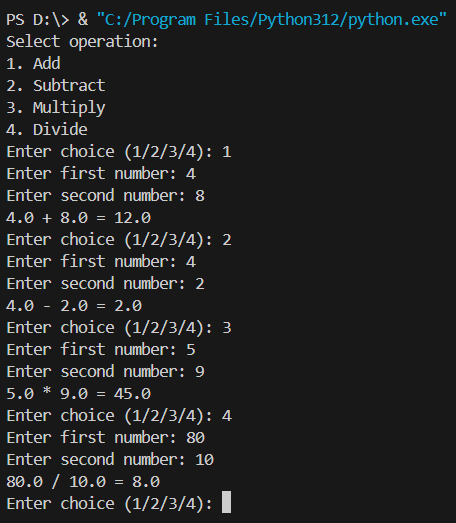
print(f"{num1} / {num2} = {div(num1, num2)}")

except ValueError:

print("Invalid input. Please enter a valid number.")

else:

print("Invalid input. Please enter a valid choice.")



**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):

if units <= 200:

amount = units \* 0.50

elif units <= 400:

amount = 100 + (units - 200) \* 0.65

elif units <= 600:

amount = 230 + (units - 400) \* 0.80

else:

amount = 390 + (units - 600)

if amount > 400:

surcharge = 0.15 \* amount

amount += surcharge

amount = max(amount, 100)

return amount

def main():

try:

units\_consumed = float(input("Enter the consumption units: "))

except ValueError:

print("Invalid input. Please enter a valid number.")

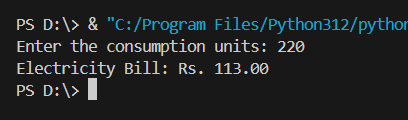
return

total\_bill = calculate\_electricity\_bill(units\_consumed)

print(f"Electricity Bill: Rs. {total\_bill:.2f}")

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

main()



**1.3. Print the pyramid of numbers using for loops.**

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

k = 0

count=0

count1=0

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

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

print(" ", end="")

count+=1

while k!=((2\*i)-1):

if count<=rows-1:

print(i+k, end=" ")

count+=1

else:

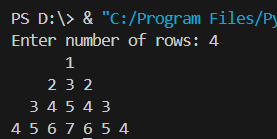
count1+=1

print(i+k-(2\*count1), end=" ")

k += 1

count1 = count = k = 0

print()



**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.**

def find\_numbers\_and\_sum():

start = 101

end = 199

divisible\_by = 7

numbers = [i for i in range(start, end + 1) if i % divisible\_by == 0]

count = len(numbers)

total\_sum = sum(numbers)

return count, total\_sum

def main():

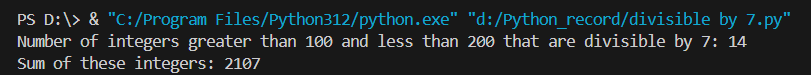
count, total\_sum = find\_numbers\_and\_sum()

print(f"Number of integers greater than 100 and less than 200 that are divisible by 7: {count}")

print(f"Sum of these integers: {total\_sum}")

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

main()



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

def sum\_numbers(n):

if n == 0:

return 0

else:

return n + sum\_numbers(n - 1)

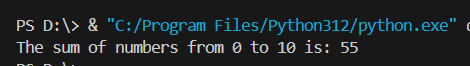
def main():

result = sum\_numbers(10)

print(f"The sum of numbers from 0 to 10 is: {result}")

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

main()



**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(number):

return str(number) == str(number)[::-1]

def reverse\_number(number):

return int(str(number)[::-1])

def find\_palindrome(number):

while not is\_palindrome(number):

reversed\_number = reverse\_number(number)

number += reversed\_number

print(f"Intermediate sum: {number}")

return number

def main():

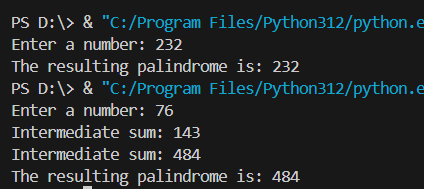
number = int(input("Enter a number: "))

palindrome = find\_palindrome(number)

print(f"The resulting palindrome is: {palindrome}")

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

main()



**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 Lettersdef is\_substring(main\_string, sub\_string):

return sub\_string in main\_string

def count\_occurrences(main\_string, char):

return main\_string.count(char)

def replace\_substring(main\_string, old\_substring, new\_substring):

return main\_string.replace(old\_substring, new\_substring)

def convert\_to\_capital(main\_string):

return main\_string.upper()

def menu():

print("Menu:")

print("1. Check if the String is a Substring of Another String")

print("2. Count Occurrences of Character")

print("3. Replace a Substring with Another Substring")

print("4. Convert to Capital Letters")

print("5. Exit")

def main():

while True:

menu()

choice = input("Enter your choice (1/2/3/4/5): ")

if choice == '1':

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

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

if is\_substring(main\_string, sub\_string):

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

else:

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

elif choice == '2':

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

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

count = count\_occurrences(main\_string, char)

print(f"The character '{char}' occurs {count} times in '{main\_string}'")

elif choice == '3':

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

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

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

result = replace\_substring(main\_string, old\_substring, new\_substring)

print(f"The new string is: '{result}'")

elif choice == '4':

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

result = convert\_to\_capital(main\_string)

print(f"The string in capital letters is: '{result}'")

elif choice == '5':

print("Exiting the program.")

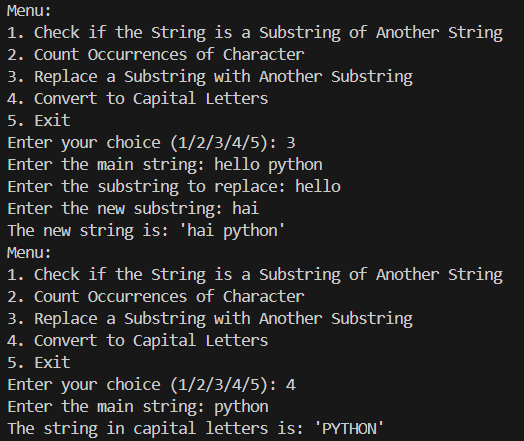
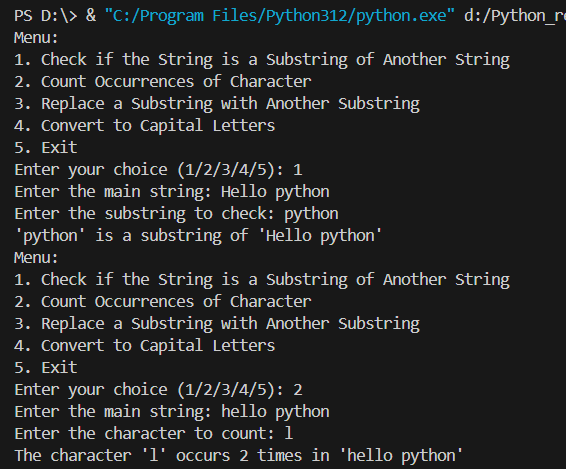
break

else:

print("Invalid choice. Please enter a valid option.")

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

main()



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

**calculated in a dictionary.**

factorial\_cache = {}

def factorial(n):

if n in factorial\_cache:

return factorial\_cache[n]

if n == 0 or n == 1:

factorial\_cache[n] = 1

else:

factorial\_cache[n] = n \* factorial(n - 1)

return factorial\_cache[n]

def main():

while True:

try:

number = int(input("Enter a number to find its factorial (or a negative number to exit): "))

if number < 0:

print("Exiting the program.")

break

result = factorial(number)

print(f"The factorial of {number} is: {result}")

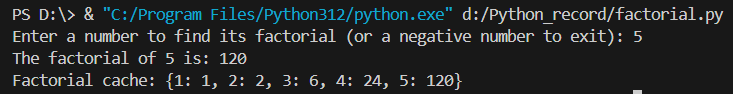
print(f"Factorial cache: {factorial\_cache}")

except ValueError:

print("Invalid input. Please enter a valid number.")

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

main()



**1.9. Perform various set operations**

**1.9.1. Set Union**

**1.9.2. Set Intersection**

**1.9.3. Set Difference**

def set\_operations(set1, set2):

while True:

print("Set Operations Menu:")

print("1. Set Union")

print("2. Set Intersection")

print("3. Set Difference")

print("4. Exit")

choice = input("Enter your choice (1/2/3/4): ")

if choice == '1':

result = set1.union(set2)

print(f"Union of sets: {result}")

elif choice == '2':

result = set1.intersection(set2)

print(f"Intersection of sets: {result}")

elif choice == '3':

result = set1.difference(set2)

print(f"Difference of sets (set1 - set2): {result}")

elif choice == '4':

print("Exiting the program.")

break

else:

print("Invalid choice. Please enter a valid option.")

def main():

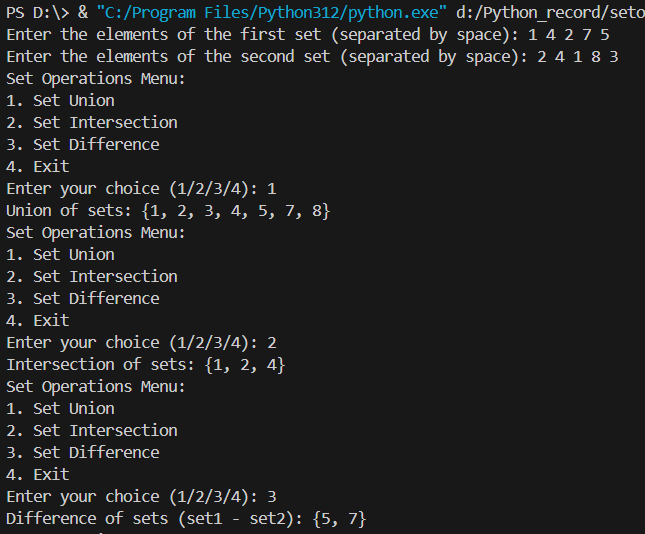
set1 = set(map(int, input("Enter the elements of the first set (separated by space): ").split()))

set2 = set(map(int, input("Enter the elements of the second set (separated by space): ").split()))

set\_operations(set1, set2)

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

main()



**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 get\_student\_details():

students = []

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

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

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

name = input("Name: ")

roll\_no = input("Roll No.: ")

total\_mark = float(input("Total Marks: "))

student = {'name': name, 'roll\_no': roll\_no, 'total\_mark': total\_mark}

students.append(student)

return students

def find\_student\_with\_highest\_mark(students):

if not students:

return None

highest\_student = max(students, key=lambda x: x['total\_mark'])

return highest\_student

def main():

students = get\_student\_details()

highest\_student = find\_student\_with\_highest\_mark(students)

if highest\_student:

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

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

print(f"Roll No.: {highest\_student['roll\_no']}")

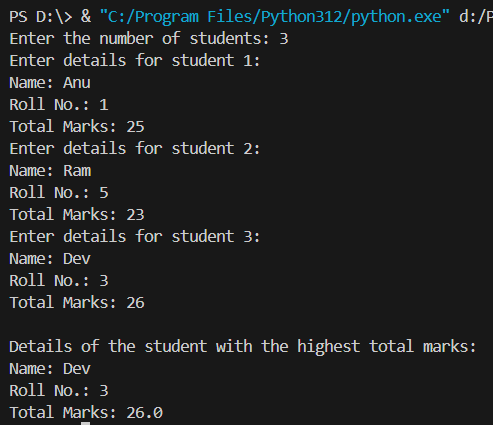
print(f"Total Marks: {highest\_student['total\_mark']}")

else:

print("No students found.")

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

main()



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

**line.**

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

try:

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

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

for line in f\_source:

f\_dest.write(line)

print(f"Contents of '{source\_file}' copied to '{destination\_file}' successfully.")

except FileNotFoundError:

print(f"Error: One or both files not found.")

except Exception as e:

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

def main():

try:

source\_file = input("Enter the source file name: ")

destination\_file = input("Enter the destination file name: ")

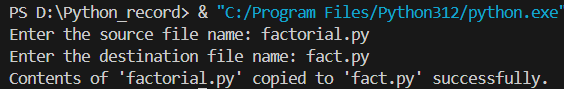
copy\_file(source\_file, destination\_file)

except Exception as e:

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

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

main()



**1.12. Use the OS module to perform**

**1.12.1. Create a directory**

import os

def create\_directory(directory\_name):

try:

os.mkdir(directory\_name)

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

except FileExistsError:

print(f"Directory '{directory\_name}' already exists.")

except Exception as e:

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

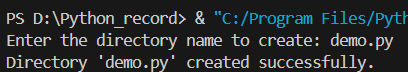
def main():

directory\_name = input("Enter the directory name to create: ")

create\_directory(directory\_name)

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

main()



**1.12.2. Directory Listing**

import os

def directory\_listing(directory\_path):

try:

print(f"Listing files and directories in '{directory\_path}':")

for item in os.listdir(directory\_path):

print(item)

except FileNotFoundError:

print(f"Directory '{directory\_path}' not found.")

except Exception as e:

print(f"Error occurred while listing directory '{directory\_path}': {e}")

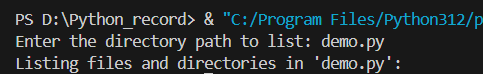
def main():

directory\_path = input("Enter the directory path to list: ")

directory\_listing(directory\_path)

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

main()



**1.12.3. Search for “.py” files.**

import os

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

try:

print(f"Searching for '.py' files in '{directory\_path}':")

for root, dirs, files in os.walk(directory\_path):

for file in files:

if file.endswith(".py"):

print(os.path.join(root, file))

except FileNotFoundError:

print(f"Directory '{directory\_path}' not found.")

except Exception as e:

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

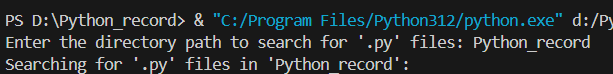
def main():

directory\_path = input("Enter the directory path to search for '.py' files: ")

search\_py\_files(directory\_path)

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

main()



**1.12.4. Remove a particular file.**

import os

def remove\_file(file\_path):

try:

# Check file attributes

print(f"File attributes for '{file\_path}':")

os.system(f'attrib "{file\_path}"')

# Try to remove the file

os.remove(file\_path)

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

except FileNotFoundError:

print(f"File '{file\_path}' not found.")

except Exception as e:

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

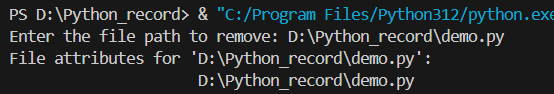
def main():

file\_path = input("Enter the file path to remove: ")

remove\_file(file\_path)

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

main()



**1.13. Create a simple banking application by using inheritance.**

class BankAccount:

def \_\_init\_\_(self, account\_number, balance=0):

self.account\_number = account\_number

self.balance = balance

def deposit(self, amount):

if amount > 0:

self.balance += amount

print(f"Deposited {amount} into account {self.account\_number}. New balance: {self.balance}")

else:

print("Invalid amount. Deposit failed.")

def withdraw(self, amount):

if 0 < amount <= self.balance:

self.balance -= amount

print(f"Withdrew {amount} from account {self.account\_number}. New balance: {self.balance}")

else:

print("Insufficient funds. Withdrawal failed.")

def display\_balance(self):

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

class SavingsAccount(BankAccount):

def \_\_init\_\_(self, account\_number, balance=0, interest\_rate=0.01):

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

self.interest\_rate = interest\_rate

def add\_interest(self):

interest\_amount = self.balance \* self.interest\_rate

self.balance += interest\_amount

print(f"Interest added to account {self.account\_number}. New balance: {self.balance}")

class CurrentAccount(BankAccount):

def \_\_init\_\_(self, account\_number, balance=0, overdraft\_limit=1000):

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

self.overdraft\_limit = overdraft\_limit

def withdraw(self, amount):

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

self.balance -= amount

print(f"Withdrew {amount} from account {self.account\_number}. New balance: {self.balance}")

else:

print("Insufficient funds. Withdrawal failed.")

def display\_balance(self):

print(f"Current Account {self.account\_number} balance: {self.balance}")

def main():

savings\_acc = SavingsAccount("SAV123", 5000, 0.02)

savings\_acc.display\_balance()

savings\_acc.deposit(2000)

savings\_acc.add\_interest()

savings\_acc.withdraw(1500)

savings\_acc.display\_balance()

current\_acc = CurrentAccount("CUR456", 10000, 2000)

current\_acc.display\_balance()

current\_acc.deposit(3000)

current\_acc.withdraw(12000)

current\_acc.display\_balance()

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

main()

