**Classes**

1. Write a python class to convert an integer into a roman numeral and viceversa

class RomanNumerals:

def \_\_init\_\_(self):

self.roman\_numerals = {

1000: 'M',

900: 'CM',

500: 'D',

400: 'CD',

100: 'C',

90: 'XC',

50: 'L',

40: 'XL',

10: 'X',

9: 'IX',

5: 'V',

4: 'IV',

1: 'I'

}

def to\_roman(self, number):

roman\_numeral = ''

for value, symbol in self.roman\_numerals.items():

while number >= value:

roman\_numeral += symbol

number -= value

return roman\_numeral

def to\_integer(self, roman\_numeral):

number = 0

for i in range(len(roman\_numeral)):

if i + 1 < len(roman\_numeral) and self.roman\_numerals[roman\_numeral[i]] < self.roman\_numerals[roman\_numeral[i + 1]]:

number -= self.roman\_numerals[roman\_numeral[i]]

else:

number += self.roman\_numerals[roman\_numeral[i]]

return number

if \_\_name\_\_ == '\_\_main\_\_':

roman\_numerals = RomanNumerals()

number = int(input("Enter an integer: "))

roman\_numeral = roman\_numerals.to\_roman(number)

print(f"{number} in Roman numerals is {roman\_numeral}")

number = roman\_numerals.to\_integer(roman\_numeral)

print(f"{roman\_numeral} in integers is {number}")

1. Write a Python class to find validity of a string of parentheses, '(', ')', '{', '}', '[' and ']. These brackets must be close in the correct order, for example "()" and "()[]{}" are valid but "[)", "({[)]" and "{{{" are invalid.

class ParenthesesValidator:

def \_\_init\_\_(self):

self.opening\_brackets = ['(', '{', '[']

self.closing\_brackets = [')', '}', ']']

self.bracket\_map = {

'(': ')',

'{': '}',

'[': ']'

}

def is\_valid(self, string):

stack = []

for character in string:

if character in self.opening\_brackets:

stack.append(character)

elif character in self.closing\_brackets:

if not stack:

return False

if character != self.bracket\_map[stack.pop()]:

return False

return not stack

1. Write a Python class to get all possible unique subsets from a set of distinct integers Input : [4, 5, 6] Output : [[], [6], [5], [5, 6], [4], [4, 6], [4, 5], [4, 5, 6]]

ANS:

class SubsetGenerator: def init (self):

self.subsets = []

def generate\_subsets(self, nums): self.\_backtrack(nums, 0, []) return self.subsets

def \_backtrack(self, nums, start, path): self.subsets.append(path[:])

for i in range(start, len(nums)): path.append(nums[i]) self.\_backtrack(nums, i + 1, path) path.pop()

subset\_generator = SubsetGenerator() input\_set = [4, 5, 6]

result = subset\_generator.generate\_subsets(input\_set) print("All possible unique subsets:")

print(result)

1. Write a Python class to find a pair of elements (indices of the two numbers) from a given array whose sum equals a specific target number. Note: There will be one solution for each input and do not use the same element twice. Input: numbers= [90, 20,10,40,50,60,70], target=50 Output: 3, 4

ANS:

class TwoSumFinder:

def find\_indices(self, numbers, target): num\_index\_mapping = {}

for i, num in enumerate(numbers): complement = target - num

if complement in num\_index\_mapping:

return num\_index\_mapping[complement], i num\_index\_mapping[num] = i

return None

two\_sum\_finder = TwoSumFinder() numbers = [90, 20, 10, 40, 50, 60, 70]

target = 50

indices = two\_sum\_finder.find\_indices(numbers, target)

if indices:

print(f"The pair indices that sum to {target} are: {indices[0]}, {indices[1]}") else:

print("No such pair found.")

1. Write a Python class to find the three elements that sum to zero from a set of n real numbers. Input array : [-25, -10, -7, -3, 2, 4, 8, 10] Output : [[-10, 2, 8], [-7, -3, 10]]

class ThreeSum:

def \_\_init\_\_(self):

self.triplets = []

def find\_triplets(self, nums):

"""

Finds all triplets in the given array that sum to zero.

Parameters:

nums: A list of numbers.

Returns:

A list of triplets.

"""

# Sort the array in ascending order.

nums.sort()

# Iterate over the array.

for i in range(len(nums) - 2):

# Skip duplicate elements.

if i > 0 and nums[i] == nums[i - 1]:

continue

# Set the left and right pointers.

left = i + 1

right = len(nums) - 1

# Move the pointers towards each other until they meet.

while left < right:

# Check if the sum of the three elements is zero.

sum = nums[i] + nums[left] + nums[right]

if sum == 0:

# Add the triplet to the list.

self.triplets.append([nums[i], nums[left], nums[right]])

# Skip duplicate elements.

while left < right and nums[left] == nums[left + 1]:

left += 1

while left < right and nums[right] == nums[right - 1]:

right -= 1

# Move the pointers towards each other.

left += 1

right -= 1

elif sum < 0:

# Move the left pointer towards the right.

left += 1

else:

# Move the right pointer towards the left.

right -= 1

return self.triplets

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

nums = [-25, -10, -7, -3, 2, 4, 8, 10]

triplets = ThreeSum().find\_triplets(nums)

print(triplets) # Output: [[-10, 2, 8], [-7, -3, 10]]

1. Write a Python class to implement pow(x, n) ANS:

class PowerCalculator: def pow(self, x, n):

if n == 0: return 1

if n < 0:

return 1 / self.pow(x, -n) half\_pow = self.pow(x, n // 2) if n % 2 == 0:

return half\_pow \* half\_pow

else:

return half\_pow \* half\_pow \* x power\_calculator = PowerCalculator()

x = float(input("Enter the value of x: ")) n = int(input("Enter the value of n: ")) result = power\_calculator.pow(x, n)

print("{x} raised to the power of {n} is: {result}".format(x=x, n=n, result=result))

1. Write a Python class to reverse a string word by word. Input string : 'hello .py' Expected Output : '.py hello' ANS:

class StringReverser:

def reverse\_words(self, s): words = s.split()

reversed\_string = " ".join(reversed(words)) return reversed\_string

string\_reverser = StringReverser() input\_string = input("Enter a string: ")

reversed\_string = string\_reverser.reverse\_words(input\_string) print("Reversed string word by word:", reversed\_string)

1. Write a python class which has 2 methods get\_string and print\_string. get\_string takes a string from the user and print\_string prints the string in reverse order.

ANS:

class StringManager: def init (self):

self.user\_string = ""

def get\_string(self):

self.user\_string = input("Enter a string: ")

def print\_string(self):

print("Reversed string:", self.user\_string[::-1])

string\_manager = StringManager() string\_manager.get\_string() string\_manager.print\_string()

1. Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.

ANS:

class Circle:

def init (self, radius): self.radius = radius

def area(self):

return 3.14159 \* self.radius\*\*2

def perimeter(self):

return 2 \* 3.14159 \* self.radius

radius = float(input("Enter the radius of the circle: ")) circle = Circle(radius)

print("Area of the circle:", circle.area()) print("Perimeter of the circle:", circle.perimeter())

1. Write a Python program to get the class name of an instance in Python. ANS:

class MyClass:

def init (self): pass

my\_instance = MyClass()

class\_name = type(my\_instance). name print("Class name of the instance:", class\_name)

**Lambda**

1. Write a Python program to create a lambda function that adds 15 to a given number passed in as an argument, also create a lambda function that multiplies argument x with argument y and prints the result.

Sample Output: 25 48

ANS:

add\_15 = lambda x: x + 15 multiply = lambda x, y: x \* y

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

x = int(input("Enter the first number for multiplication: "))

y = int(input("Enter the second number for multiplication: "))

result\_add = add\_15(number) result\_multiply = multiply(x, y)

print("Result of adding 15 to the number:", result\_add) print("Result of multiplying", x, "and", y, "is:", result\_multiply)

1. Write a Python program to sort a list of tuples using Lambda.

Original list of tuples: [('English', 88), ('Science', 90), ('Maths', 97), ('Social sciences', 82)] Sorting the List of Tuples: [('Social sciences', 82), ('English', 88), ('Science', 90), ('Maths', 97)] ANS:

n = int(input("Enter the number of tuples: ")) original\_list = []

for i in range(n):

subject = input("Enter the subject name: ") score = int(input("Enter the score: ")) original\_list.append((subject, score))

sorted\_list = sorted(original\_list, key=lambda x: x[1]) print("Sorting the List of Tuples:", sorted\_list)

1. Write a Python program to sort a list of dictionaries using Lambda.

Original list of dictionaries : [{'make': 'Nokia', 'model': 216, 'color': 'Black'}, {'make': 'Mi Max',

'model': '2', 'color': 'Gold'}, {'make': 'Samsung', 'model': 7, 'color': 'Blue'}]

Sorting the List of dictionaries : [{'make': 'Nokia', 'model': 216, 'color': 'Black'}, {'make': 'Samsung', 'model': 7, 'color': 'Blue'}, {'make': 'Mi Max', 'model': '2', 'color': 'Gold'}] ANS:

original\_list = [{'make': 'Nokia', 'model': 216, 'color': 'Black'},

{'make': 'Mi Max', 'model': '2', 'color': 'Gold'},

{'make': 'Samsung', 'model': 7, 'color': 'Blue'}]

sorted\_list = sorted(original\_list, key=lambda x: x['make']) print("Sorting the List of dictionaries:", sorted\_list)

4) Write a Python program to find if a given string starts with a given character using Lambda.

def starts\_with(string, char):

return string[0] == char

print(starts\_with("Hello", "H")) # Output: True

print(starts\_with("Hello", "h")) # Output: False

1. Write a Python program to check whether a given string is a number or not using Lambda. ANS:

is\_number = lambda s: s.isdigit() input\_string = input("Enter a string: ")

if is\_number(input\_string):

print("The string '{}' is a number.".format(input\_string)) else:

print("The string '{}' is not a number.".format(input\_string))

1. Write a Python program to find numbers divisible by nineteen or thirteen from a list of numbers using Lambda

Original list: [19, 65, 57, 39, 152, 639, 121, 44, 90, 190]

Numbers of the above list divisible by nineteen or thirteen: [19, 65, 57, 39, 152, 190] ANS:

original\_list = [19, 65, 57, 39, 152, 639, 121, 44, 90, 190]

divisible\_by\_nineteen\_or\_thirteen = lambda x: x % 19 == 0 or x % 13 == 0 result\_list = list(filter(divisible\_by\_nineteen\_or\_thirteen, original\_list)) print("Numbers of the above list divisible by nineteen or thirteen:", result\_list)

1. Write a Python program to sort a given matrix in ascending order according to the sum of its rows using lambda.

Original Matrix: [[1, 2, 3], [2, 4, 5], [1, 1, 1]]

Sort the said matrix in ascending order according to the sum of its rows [[1, 1, 1], [1, 2, 3], [2, 4,

5]]

Original Matrix: [[1, 2, 3], [-2, 4, -5], [1, -1, 1]]

Sort the said matrix in ascending order according to the sum of its rows [[-2, 4, -5], [1, -1, 1], [1,

2, 3]] ANS:

def sort\_matrix\_by\_row\_sum(matrix):

sorted\_matrix = sorted(matrix, key=lambda row: sum(row)) return sorted\_matrix

original\_matrix1 = [[1, 2, 3], [2, 4, 5], [1, 1, 1]]

original\_matrix2 = [[1, 2, 3], [-2, 4, -5], [1, -1, 1]]

sorted\_matrix1 = sort\_matrix\_by\_row\_sum(original\_matrix1) sorted\_matrix2 = sort\_matrix\_by\_row\_sum(original\_matrix2)

print("Original Matrix 1:", original\_matrix1)

print("Sort the matrix 1 in ascending order according to the sum of its rows:", sorted\_matrix1)

print("\nOriginal Matrix 2:", original\_matrix2)

print("Sort the matrix 2 in ascending order according to the sum of its rows:", sorted\_matrix2)

1. Write a Python program to check whether a given string contains a capital letter, a lower case letter, a number and a minimum length using lambda. Minimum length : 10 input string: PaceWisd0m o/p: valid string
2. Write a Python program to find the elements of a given list of strings that contain specific substring using lambda.

Original list: ['red', 'black', 'white', 'green', 'orange']

Substring to search: ack Elements of the said list that contain specific substring: ['black'] Substring to search: abc Elements of the said list that contain specific substring: []

ANS:

original\_list = ['red', 'black', 'white', 'green', 'orange'] contains\_substring = lambda s, substring: substring in s substring\_to\_search = input("Substring to search: ")

filtered\_list = list(filter(lambda s: contains\_substring(s, substring\_to\_search), original\_list)) print("Elements of the list that contain the specific substring:", filtered\_list)

1. Write a Python program to sort a given mixed list of integers and strings using lambda. Numbers must be sorted before strings.

Original list: [19, 'red', 12, 'green', 'blue', 10, 'white', 'green', 1]

Sort the said mixed list of integers and strings: [1, 10, 12, 19, 'blue', 'green', 'green', 'red', 'white'] ANS:

original\_list = [19, 'red', 12, 'green', 'blue', 10, 'white', 'green', 1] sort\_order = lambda x: (isinstance(x, int), x)

sorted\_list = sorted(original\_list, key=sort\_order)

print("Sort the mixed list of integers and strings:", sorted\_list