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

class RomanNumeralConverter: 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 int\_to\_roman(self, num):

roman\_numeral = ''

for value, numeral in self.roman\_numerals.items(): while num >= value:

roman\_numeral += numeral num -= value

return roman\_numeral

def roman\_to\_int(self, roman\_numeral): result = 0

for value, numeral in reversed(list(self.roman\_numerals.items())): while roman\_numeral.startswith(numeral):

result += value

roman\_numeral = roman\_numeral[len(numeral):] return result

# 2.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 BracketValidator:**

def init (self, input\_string): self.input\_string = input\_string

def is\_valid(self): stack = []

for char in self.input\_string:

if char in ["(", "{", "["]:

stack.append(char) else:

if not stack: return False

current\_char = stack.pop() if current\_char == "(":

if char != ")": return False

if current\_char == "{":

if char != "}": return False

if current\_char == "[": if char != "]":

return False

if stack:

return False

return True

## 3.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]]**

class SubsetGenerator:

def init (self, input\_set): self.input\_set = input\_set self.subsets = []

def generate\_subsets(self): self.\_generate\_helper([], 0) return self.subsets

def \_generate\_helper(self, current\_subset, start\_index): self.subsets.append(current\_subset)

for i in range(start\_index, len(self.input\_set)):

self.\_generate\_helper(current\_subset + [self.input\_set[i]], i + 1)

## 4.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

class TwoSum:

def find\_indices(self, numbers, target): """

Finds a pair of indices whose elements' sum equals the target.

:param numbers: a list of integers

:param target: an integer

:return: a tuple of indices (i, j) such that numbers[i] + numbers[j] = target """

seen = {}

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

if complement in seen:

return seen[complement], i seen[num] = i

return None

## 5.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 find\_triplets(self, nums): """

Finds all triplets in nums that add up to zero.

:param nums: a list of integers

:return: a list of lists of integers, each inner list representing a triplet that adds up to zero

"""

nums.sort()

n = len(nums) triplets = []

for i in range(n - 2):

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

j, k = i + 1, n - 1

while j < k:

total = nums[i] + nums[j] + nums[k]

if total == 0:

triplets.append([nums[i], nums[j], nums[k]]) j += 1

k -= 1

while j < k and nums[j] == nums[j - 1]: j += 1

while j < k and nums[k] == nums[k + 1]:

k -= 1

elif total < 0: j += 1

else:

k -= 1

return triplets

## 6.Write a Python class to implement pow(x, n)

class Pow:

def my\_pow(self, x: float, n: int) -> float: """

Computes x raised to the power n.

:param x: a float, the base

:param n: an integer, the exponent

:return: a float, x raised to the power n

"""

if n == 0: return 1

elif n < 0:

return 1 / self.my\_pow(x, -n) elif n % 2 == 0:

return self.my\_pow(x \* x, n // 2) else:

return x \* self.my\_pow(x, n - 1)

## 7.Write a Python class to reverse a string word by word. Input string : 'hello .py' Expected Output : '.py hello’ **class StringReverser:**

def reverse\_words(self, s: str) -> str: """

Reverses the words in a string.

:param s: a string to be reversed

:return: a string with the words reversed

"""

# Split the string into words words = s.split()

# Reverse the order of the words reversed\_words = words[::-1]

# Join the words back together into a single string

reversed\_string = ' '.join(reversed\_words)

return reversed\_string

## 8.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 class StringReverser:

def init (self): self.string = ""

def get\_string(self): """

Takes a string input from the user and stores it in the class instance variable `string`.

"""

self.string = input("Enter a string: ")

def print\_string(self): """

Prints the stored string in reverse order. """

reversed\_string = self.string[::-1]

print("Reversed string:", reversed\_string)

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

class Circle:

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

def area(self): """

Computes the area of the circle.

"""

return 3.14 \* (self.radius \*\* 2)

def perimeter(self): """

Computes the perimeter (circumference) of the circle. """

return 2 \* 3.14 \* self.radius

## 10.Write a Python program to get the class name of an instance in Python

class MyClass: pass

obj = MyClass()

# Get the class of the object using type() class\_name = type(obj). name

print("Class name:", 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 print the result.

**Sample Output: 25 48**

# Create a lambda function that adds 15 to a number add\_15 = lambda x: x + 15

# Create a lambda function that multiplies two numbers

multiply = lambda x, y: x \* y

# Test the lambda functions x = 10

y = 4

# Add 15 to x using the add\_15 function result1 = add\_15(x)

# Multiply x and y using the multiply function result2 = multiply(x, y)

print(result1, result2)

## 2.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)]

# Define the original list of tuples

tuples\_list = [('English', 88), ('Science', 90), ('Maths', 97), ('Social sciences', 82)]

# Sort the list of tuples using a lambda function sorted\_tuples\_list = sorted(tuples\_list, key=lambda x: x[1])

# Print the sorted list of tuples

print("Sorting the List of Tuples:", sorted\_tuples\_list)

## 3.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’}]

# Define the original list of dictionaries dict\_list = [

{'make': 'Nokia', 'model': 216, 'color': 'Black'},

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

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

]

# Sort the list of dictionaries using a lambda function sorted\_dict\_list = sorted(dict\_list, key=lambda x: x['make'])

# Print the sorted list of dictionaries

print("Sorting the List of dictionaries:", sorted\_dict\_list)

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

**starts\_with = lambda string, char: string.startswith(char)**

# Example usage string = "hello world" char = "h"

print(starts\_with(string, char)) # Output: True

char = "w"

print(starts\_with(string, char)) # Output: False

## 5.Write a Python program to check whether a given string is number or not using Lambda

is\_number = lambda s: s.replace('.','',1).isdigit() # Example usage

string1 = "123"

string2 = "3.14"

string3 = "-42"

string4 = "not a number"

print(is\_number(string1)) # Output: True print(is\_number(string2)) # Output: True print(is\_number(string3)) # Output: True print(is\_number(string4)) # Output: False

## 6.Write a Python program to find numbers divisible by nineteen or thirteen from a

**list of numbers using Lambda**

## Orginal 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]

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

divisible\_by\_19\_or\_13 = list(filter(lambda x: x % 19 == 0 or x % 13 == 0, original\_list)) print("Original list:", original\_list)

print("Numbers of the above list divisible by nineteen or thirteen:", divisible\_by\_19\_or\_13)

## 7.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]]

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

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

sorted\_matrix1 = sorted(original\_matrix1, key=lambda x: sum(x)) sorted\_matrix2 = sorted(original\_matrix2, key=lambda x: sum(x))

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

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

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

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

**8.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**

input\_string = "PaceWisd0m"

check\_valid = lambda s: any(c.isupper() for c in s) and \ any(c.islower() for c in s) and \ any(c.isdigit() for c in s) and \

len(s) >= 10

if check\_valid(input\_string): print("Valid string")

else:

print("Invalid string")

## 9.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: []

original\_list = ['red', 'black', 'white', 'green', 'orange'] substring1 = 'ack'

substring2 = 'abc'

contains\_substring = lambda s, sub: sub in s

result1 = list(filter(lambda x: contains\_substring(x, substring1), original\_list))

result2 = list(filter(lambda x: contains\_substring(x, substring2), original\_list))

print("Original list:", original\_list)

print(f"Elements of the list that contain '{substring1}':", result1) print(f"Elements of the list that contain '{substring2}':", result2)

## 10.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']**

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

print("Original list:", original\_list) print("Sorted list:", sorted\_list)