Classes:

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

class RomanConverter:

def int\_to\_roman(self, num):

val = [1000, 900, 500, 400,100, 90, 50, 40,10, 9, 5, 4,1]

syms = ["M", "CM", "D", "CD","C", "XC", "L", "XL","X", "IX", "V", "IV","I"]

roman\_numeral = ''

i = 0

while num > 0:

for \_ in range(num // val[i]):

roman\_numeral += syms[i]

num -= val[i]

i += 1

return roman\_numeral

def roman\_to\_int(self, s):

val\_dict = {'I': 1, 'V': 5, 'X': 10,'L': 50, 'C': 100, 'D': 500,'M': 1000}

total = 0

prev\_value = 0

for c in s[::-1]:

value = val\_dict[c]

if value < prev\_value:

total -= value

else:

total += value

prev\_value = value

return total

converter = RomanConverter()

print(converter.int\_to\_roman(1987))

print(converter.roman\_to\_int('MCMXCIV'))

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 ParenthesesValidator:

def is\_valid(self, s):

stack = []

brackets = {'(': ')', '[': ']', '{': '}'}

for char in s:

if char in brackets:

stack.append(char)

elif char in brackets.values():

if not stack or brackets[stack.pop()] != char:

return False

return not stack

validator = ParenthesesValidator()

print(validator.is\_valid("()[]{}"))

print(validator.is\_valid("[({[)]"))

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 generate\_subsets(self, nums):

def backtrack(start, curr\_subset):

res.append(curr\_subset[:])

for i in range(start, len(nums)):

curr\_subset.append(nums[i])

backtrack(i + 1, curr\_subset)

curr\_subset.pop()

res = []

backtrack(0, [])

return res

generator = SubsetGenerator()

print(generator.generate\_subsets([4, 5, 6]))

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 PairSumFinder:

def find\_pair(self, numbers, target):

num\_indices = {}

for idx, num in enumerate(numbers):

if target - num in num\_indices:

return [num\_indices[target - num], idx]

num\_indices[num] = idx

finder = PairSumFinder()

print(finder.find\_pair([90, 20, 10, 40, 50, 60, 70], 50))

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 ThreeSumFinder:

def find\_three\_sum(self, nums):

res = []

nums.sort()

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

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

continue

left, right = i + 1, len(nums) - 1

while left < right:

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

if total < 0:

left += 1

elif total > 0:

right -= 1

else:

res.append([nums[i], nums[left], nums[right]])

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

left += 1

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

right -= 1

left += 1

right -= 1

return res

finder = ThreeSumFinder()

print(finder.find\_three\_sum([-25, -10, -7, -3, 2, 4, 8, 10]))

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

class PowerCalculator:

def pow(self, x, n):

if x==0 or x==1 or n==1:

return x

if x==-1:

if n%2 ==0:

return 1

else:

return -1

if n==0:

return 1

if n<0:

return 1/self.pow(x,-n)

val = self.pow(x,n//2)

if n%2 ==0:

return val\*val

return val\*val\*x

calculator = PowerCalculator()

print(calculator.pow(2, 5))

OR

class PowerCalculator:

def pow(self, x, n):

return x \*\* n

calculator = PowerCalculator()

print(calculator.pow(2, 5))

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):

return ' '.join(reversed(s.split()))

reverser = StringReverser()

print(reverser.reverse\_words('hello .py'))

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 get\_string(self):

return input("Enter a string: ")

def print\_string\_reverse(self, s):

print(s[::-1])

reverser = StringReverser()

input\_string = reverser.get\_string()

reverser.print\_string\_reverse(input\_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.

import math

class Circle:

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

self.radius = radius

def compute\_area(self):

return math.pi \* self.radius \*\* 2

def compute\_perimeter(self):

return 2 \* math.pi \* self.radius

circle = Circle(5)

print(circle.compute\_area())

print(circle.compute\_perimeter())

OR

class Circle():

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

self.radius = r

def area(self):

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

def perimeter(self):

return 2\*self.radius\*3.14

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

def get\_instance (instance):

return instance.\_\_class\_\_.\_\_name\_\_

class EG:

pass

obj = EG()

print(get\_instance (obj))

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

add\_15 = lambda x: x + 15

multiply = lambda x, y: x \* y

num = 10

print(add\_15(num))

print(multiply(6, 8))

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)]

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

sorted\_data = sorted(data, key=lambda x: x[1])

print(sorted\_data)

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'}]

data = [{'make': 'Nokia', 'model': 216, 'color': 'Black'}, {'make': 'Mi Max', 'model': '2', 'color': 'Gold'}, {'make': 'Samsung', 'model': 7, 'color': 'Blue'}]

sorted\_data = sorted(data, key=lambda x: x['model'])

print(sorted\_data)

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

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

string = "Shreesha"

char = "S"

print(starts\_with(string, char))

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

is\_number = lambda s: s.replace(".", "").isdigit()

string = "123.45"

print(is\_number(string))

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]

nums = [19, 65, 57, 39, 152, 639, 121, 44, 90, 190]

divisible\_nums = list(filter(lambda x: x % 19 == 0 or x % 13 == 0, nums))

print(divisible\_nums)

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]]

matrix = [[1, 2, 3], [2, 4, 5], [1, 1, 1]]

matrix.sort(key=lambda x:sum(x))

print(matrix)

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

check\_string = lambda s: any(cond(s) for cond in [str.isupper, str.islower, str.isdigit]) and len(s) >= 10

input\_string = "PaceWisd0m"

print(check\_string(input\_string))

OR

check\_string = lambda s: True if len(s) >= 10 and any(x.isupper() for x in s) and any(x.islower() for x in s) and any(x.isdigit() for x in s) else False

print(check\_string('PaceWisd0m'))

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: []

list1 = ['red', 'black', 'white', 'green', 'orange']

print(list(filter(lambda word: 'ack' in word, list1)))

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']

mixed\_list = [19, 'red', 12, 'green', 'blue', 10, 'white', 'green', 1]

mixed\_list.sort(key=lambda x: (isinstance(x, str), x))

print(mixed\_list)