1)

class Roman\_converter:

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\_num = ' '

i = 0

while num > 0:

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

roman\_num += syms[i]

num -= val[i]

i += 1

return roman\_num

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

roman\_numerals = {

'I': 1, 'V': 5, 'X': 10, 'L': 50,

'C': 100, 'D': 500, 'M': 1000

}

num = 0

prev\_value = 0

for numeral in s[::-1]:

if roman\_numerals[numeral] >= prev\_value:

num += roman\_numerals[numeral]

else:

num -= roman\_numerals[numeral]

prev\_value = roman\_numerals[numeral]

return num

converter = Roman\_converter()

print(converter.int\_to\_roman(3549))

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

2) Validity of Parentheses

class ParenValidator:

def is\_valid(self, s):

stack = []

mapping = {")": "(", "}": "{", "]": "["}

for char in s:

if char in mapping:

top\_element = stack.pop() if stack else '#'

if mapping[char] != top\_element:

return False

else:

stack.append(char)

return not stack

validator = ParenValidator()

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

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

3)

from itertools import combinations

class SubsetGenerator:

def generate\_subsets(self, nums):

subsets = []

for i in range(len(nums) + 1):

subsets.extend(list(combinations(nums, i)))

return [list(subset) for subset in subsets]

generator = SubsetGenerator()

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

4)

class PairFinder:

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

num\_indices = {}

for index, num in enumerate(nums):

complement = target - num

if complement in num\_indices:

return [num\_indices[complement], index]

num\_indices[num] = index

finder = PairFinder()

numbers = [90, 20, 10, 40, 50, 60, 70]

target = 50

print(finder.find\_pair(numbers, target)

5)

class ThreeSum:

def three\_sum(self, nums):

nums.sort()

result = []

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

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

while left < right:

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

if total == 0:

result.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

elif total < 0:

left += 1

else:

right -= 1

return result

three\_sum = ThreeSum()

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

print(three\_sum.three\_sum(nums)) # Output: [[-10, 2, 8], [-7, -3, 10]]

6)

class PowerCalculator:

def pow(self, x, n):

return x \*\* n

calculator = PowerCalculator()

print(calculator.pow(2, 3))

7)

class StringReverser:

def reverse\_words(self, s):

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

reverser = StringReverser()

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

8)

class StringManipulator:

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

self.string = ""

def get\_string(self):

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

def print\_string(self):

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

manipulator = StringManipulator()

manipulator.get\_string()

manipulator.print\_string()

9

class Circle:

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

self.radius = radius

def area(self):

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

def perimeter(self):

return 2 \* 3.14 \* self.radius

circle = Circle(5)

print("Area:", circle.area())

print("Perimeter:", circle.perimeter())

10

class ClassNameFinder:

def get\_class\_name(self, instance):

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

finder = ClassNameFinder()

circle = Circle(5)

print(finder.get\_class\_name(circle))

Lambda Functions:

1)

addition = lambda x: x + 15

multiplication = lambda x, y: x \* y

print(addition(10)) # Output: 25

print(multiplication(4, 12)) # Output: 48