Classes:

1. To convert the integers to roman numerals and vice versa

```
class Roman:
    def int_to_roman(self, num):
        val = [
            1000, 900, 500, 400,
            100, 90, 50, 40,
            10, 9, 5, 4,
            1
            ]
        syb = [
            "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 += syb[i]
                num -= val[i]
            i += 1
        return roman_num

    def roman_to_int(self, s):
        roman_vals = {
            'I': 1, 'V': 5, 'X': 10, 'L': 50,
            'C': 100, 'D': 500, 'M': 1000
        }
        num = 0
        prev_val = 0
        for char in s[::-1]:
            val = roman_vals[char]
            if val >= prev_val:
                num += val
            else:
                num -= val
            prev_val = val
        return num

converter = Roman()
print(converter.int_to_roman(1984))
```

```
print(converter.roman_to_int("MCMLXXXIV"))
```

2. To validity of a string of parenthesis

```
class ParenthesisValidator:
    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 = ParenthesisValidator()
print(validator.is_valid("()"))
print(validator.is_valid("()[]{}"))
print(validator.is_valid("(]"))
print(validator.is_valid("({[)]"))
print(validator.is_valid("{{{"))
```

3. To get all possible subsets

```
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 subsets

generator = SubsetGenerator()
print(generator.generate_subsets([4, 5, 6]))
```

4. To find a pair of elements whose sum equals to target number

```
class TwoSumFind:
    def find_two_sum(self, numbers, target):
        num_to_index = {}
        for i, num in enumerate(numbers):
```

```
            complement = target - num
            if complement in num_to_index:
                return [num_to_index[complement], i]
            num_to_index[num] = i
        return None

finder = TwoSumFind()
numbers = [90, 20, 10, 40, 50, 60, 70]
target = 50
print(finder.find_two_sum(numbers, target))
```

5. To find three elements that sum to zero

```
class ThreeSum:
    def three_sum(self, nums):
        nums.sort()
        result = []
        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:
                    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
        return result

finder = ThreeSum()
input_array = [-25, -10, -7, -3, 2, 4, 8, 10]
print(finder.three_sum(input_array))
```

6. To implement pow(x,n)

```
class PowerCalculator:
```

```
    def my_pow(self, x, n):
        if n == 0:
            return 1
        if n < 0:
            x = 1 / x
            n = -n
        return self.my_pow(x * x, n // 2) if n % 2 == 0 else x * self.my_pow(x * x, n // 2)

calculator = PowerCalculator()
print(calculator.my_pow(2, 10))
```

7. To reverse a string word by word

```
class StringReverser:
    def reverse_words(self, s):
        words = s.split()
        reversed_words = ' '.join(reversed(words))
        return reversed_words

reverser = StringReverser()
input_string = 'hello .py'
print(reverser.reverse_words(input_string))
```

8. To reverse string with methods

```
class StringManipulator:
    def __init__(self):
        self.input_string = ""

    def get_string(self):
        self.input_string = input("Enter a string: ")

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

manipulator = StringManipulator()
manipulator.get_string()
manipulator.print_string_reverse()
```

9. To find area and perimeter of circle

```
import math
class Circle:
    def __init__(self, radius):
        self.radius = radius
```

```
    def area(self):
        return math.pi * self.radius ** 2

    def perimeter(self):
        return 2 * math.pi * self.radius

circle = Circle(5)
print("Area:", circle.area())
print("Perimeter:", circle.perimeter())
```

10. To get class name of an instance

```
class ClassNameGetter:
    def get_class_name(self, instance):
        return instance.__class__.__name__

getter = ClassNameGetter()
print(getter.get_class_name(circle))  // Output: Circle
```

Lambda:

1.To create lambda function to add 15 and that multiplies x and y

```
add_15 = lambda x: x + 15

multiply = lambda x, y: x * y

print(add_15(10))

print(multiply(6, 8))

print()
```

2. To sort a list of tuples using Lambda

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

sorted_tuples = sorted(original_tuples, key=lambda x: x[1])

print("Sorting the List of Tuples:", sorted_tuples)

print()
```

3. To sort a list of dictionaries using Lambda.

```
original_dicts = [
{'make': 'Nokia', 'model': 216, 'color': 'Black'},
{'make': 'Mi Max', 'model': '2', 'color': 'Gold'},
{'make': 'Samsung', 'model': 7, 'color': 'Blue'}]
sorted_dicts = sorted(original_dicts, key=lambda x: x['make'])
print("Sorting the List of Dictionaries:", sorted_dicts)
print()
```

4. To find if a given string starts with a given character using Lambda.

```
starts_with = lambda string, char: string.startswith(char)
print(starts_with("Hello", "H"))
print(starts_with("World", "W"))
print()
```

5. To check whether a given string is number or not using Lambda.

```
is_number = lambda s: s.isnumeric()
print(is_number("12345"))
print(is_number("abc123"))
print()
```

6. To numbers divisible by nineteen or thirteen from a list of numbers using Lambda

```
original_numbers = [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_numbers))
print("Numbers divisible by nineteen or thirteen:", divisible_by_19_or_13)
print()
```

7. 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]]
sorted_matrix = sorted(original_matrix, key=lambda x: sum(x))
print("Sort the matrix in ascending order according to the sum of its rows:", sorted_matrix)
print()
```

8. To check whether a given string contains a capital letter, a lower case letter, a number and a minimum length

```
check_string = lambda s: any(cond(s) for cond in [str.islower, str.isupper, str.isdigit]) and len(s) >= 10
print(check_string("PaceWisd0m"))
```

9. To to find the elements of a given list of strings that contain specific substring using lambda.

```
original_strings = ['red', 'black', 'white', 'green', 'orange']
search_substring = lambda substr: [s for s in original_strings if substr in s]
print(search_substring("ack"))
print(search_substring("abc"))
print()
```

10.To sort a given mixed list of integers and strings using lambda. Numbers must be sorted before strings.

```
mixed_list = [19, 'red', 12, 'green', 'blue', 10, 'white', 'green', 1]
sorted_mixed_list = sorted(mixed_list, key=lambda x: (isinstance(x, int), x))
print("Sort the mixed list of integers and strings:", sorted_mixed_list)
```