## Exercise 1

**Fibonacci Code**

def fib(n):

    if n <= 1:

        return n

    else:

      return fib(n-1) + fib(n-2)

def main():

    # Iterate over 20 terms

    print('Exercise 1: Fibonacci Series')

    for i in range(21):

      # print(i)

       if i<20:

        print(fib(i), end=", ")

       else:

        print(fib(i))

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

    main()

## Exercise 2

**Greatest Common Divisor Code using Euclid’s Algorithm**

def gcd(num1, num2):

    # If condition in case number 2 is less then or equal to number1 and num1 is divisble by num2

    if num2 <= num1 and num1 % num2 == 0:

        return num2

    # In case not divisible but less than

    elif num1 < num2:

        return gcd(num2, num1)

    # Else case when num2 > num1

    return gcd(num2, num1 % num2)

def main():

    # Test function with numbers

    num1 = 90

    num2 = 18

    print(f"GCD of {num1} and {num2} is: {gcd(num1, num2)}")

    num1 = 30

    num2 = 15

    print(f"GCD of {num1} and {num2} is: {gcd(num1, num2)}")

    num1 = 60

    num2 = 90

    print(f"GCD of {num1} and {num2} is: {gcd(num1, num2)}")

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

    main()

## Exercise 3

**String Reversal Code**

def reversal(s):

    # Base case: if the string has 0 or 1 character, it's already reversed

    if len(s) <= 1:

        return s

    # Recursive case: reverse the substring excluding the first character and append the first character

    return reversal(s[1:]) + s[0]

def main():

    # Test cases

    string1 = "algorithm"

    print(f"Original string: {string1}")

    print(f"Reversed string: {reversal(string1)}")

    string2 = "programming"

    print(f"Original string: {string2}")

    print(f"Reversed string: {reversal(string2)}")

    string3 = "python is a coding language"

    print(f"Original string: {string3}")

    print(f"Reversed string: {reversal(string3)}")

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

    main()

## Exercise 4

def bar(n):

    result = 1

    for i in range(1, n):

       for j in range(1, i):

        result += 2

    return result

**Time Complexity Determination**

#### Big O Notation:

For that function, it is Big O (n ^ 2) that is quadratic because there nested loops mean to say loop is inside loop so making to iterate n size taking power of 2.

#### Contribution to Growth:

Outer loop will iterate over inside loop from 1 to n-1. Like if n is 5 then it will run 1,2,3,4 making it 4 times. Similarly, inner loop is dependent on i variable value used in outer loop. Inner loop will run from 1 to i-1 times like if is 3 then 1,2 making it 2 times. Hence, **nested loops** are making most contribution to growth of function. **Major contribution** is done by **Inner loop** because it is iterating depending on value of i.

#### Summary:

In summary, the time complexity of the bar(n) function is O(n^2), and the nested loops are basic factors taking part to the growth of the function.

## Exercise 5

**Person and Paid Advisor Class code**

class Person:

    def \_\_init\_\_(self, first\_name, last\_name):

        self.first\_name = first\_name

        self.last\_name = last\_name

    def \_\_str\_\_(self):

        return f'{self.last\_name}, {self.first\_name}'

    def get\_first\_name(self):

        return self.first\_name

    def get\_last\_name(self):

        return self.last\_name

# inherit person class

class PaidAdvisor(Person):

    # data members for pay rate by regular or special or over time

    regular\_pay\_rate = 25

    special\_pay\_rate = 50

    overtime\_pay\_rate = 1.5 \* regular\_pay\_rate

    # Constructor to intanstiate object of PairAdvisor class

    def \_\_init\_\_(self, first\_name, last\_name):

        super().\_\_init\_\_(first\_name, last\_name)

        self.hours\_worked = 0

        self.hours\_special = 0

    # Calculate Pay of Employee

    def calculate\_pay(self):

        total\_hours = self.hours\_worked + self.hours\_special

        if total\_hours <= 30:

            return total\_hours \* self.regular\_pay\_rate

        else:

            regular\_hours = min(self.hours\_worked, 30) # 30 are regular working hours

            overtime\_hours = max(total\_hours - 30, 0)

            regular\_pay = regular\_hours \* self.regular\_pay\_rate

            overtime\_pay = overtime\_hours \* self.overtime\_pay\_rate

            return regular\_pay + overtime\_pay + self.hours\_special \* self.special\_pay\_rate

    # Get pay rate

    def get\_pay\_rate(self):

        if self.hours\_worked <= 30:

            return self.regular\_pay\_rate

        else:

            return self.overtime\_pay\_rate

    # Get total hours worked

    def get\_hours\_worked(self):

        return self.hours\_worked

    # Set attributes for employee

    def set\_name\_rate\_hours(self, first\_name, last\_name, hours\_worked, hours\_special):

        self.first\_name = first\_name

        self.last\_name = last\_name

        self.hours\_worked = hours\_worked

        self.hours\_special = hours\_special

# Driver Main Function for testing classes

def main():

    # Ask user input

    first\_name = input("Enter first name: ")

    last\_name = input("Enter last name: ")

    hours\_worked = float(input("Enter hours worked: "))

    hours\_special = float(input("Enter special session hours: "))

    # Make Advisor Object

    advisor = PaidAdvisor(first\_name, last\_name)

    advisor.set\_name\_rate\_hours(first\_name, last\_name, hours\_worked, hours\_special)

    # Print Details respectively

    print("\nEmployee details:")

    print(advisor)

    print(f"Hours worked: {advisor.get\_hours\_worked()}")

    print(f"Pay rate: ${advisor.get\_pay\_rate()} per hour")

    print(f"Total pay: ${advisor.calculate\_pay()}")

# Call main function

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

    main()