Software Development - Session 2

Python programming exercises

1. A perfect number is an integer whose sum of integer divisors (excluding the number itself) add up to the number. Following are the first four perfect numbers:

6 = 1 + 2 + 3

28 = 1 + 2 + 4 + 7 + 14

496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248

8128 = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 127 + 254 + 508 + 1016 + 2032 +

4064

def is\_perfect\_number(num):

    divisors\_sum = sum(i for i in range(1, num) if num % i == 0)

    return divisors\_sum == num

def find\_perfect\_numbers(count):

    perfect\_numbers = []

num = 2  # Start checking from 2

    while len(perfect\_numbers) < count:

        if is\_perfect\_number(num):

            perfect\_numbers.append(num)

        num += 1

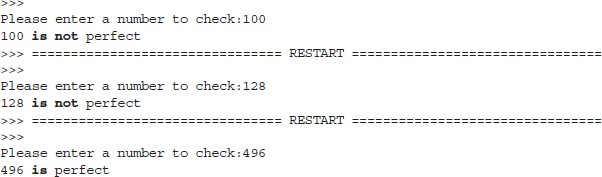
return perfect\_numbers

# Example usage to find the first four perfect numbers

first\_four\_perfect\_numbers = find\_perfect\_numbers(4)

print("First four perfect numbers:", first\_four\_perfect\_numbers)

Numbers whose sum of divisors is larger than the number are called abundant numbers. Those whose sum is less than the number are called deficient numbers. Perfect numbers have other interesting properties, but for now we are interested only in writing a program that sorts integers into one of three categories: perfect, abundant, or deficient. Write a program so that we get the following results:



def is\_perfect\_number(num):

    divisors\_sum = sum(i for i in range(1, num) if num % i == 0)

    return divisors\_sum == num

# Get user input

try:

    user\_input = int(input("Enter an integer: "))

    if is\_perfect\_number(user\_input):

        print(f"{user\_input} is a perfect")

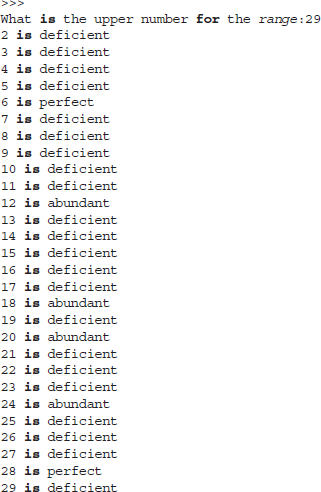
    else:

        print(f"{user\_input} is not a perfect")

except ValueError:

    print("Invalid input. Please enter a valid integer.")

1. Continuing on task 1, we would like to check a whole range of number and decide what each number is: abundant, deficient, or perfect. In this way we can look for more perfect numbers. Write a program so that we get the following results:



def classify\_number(num):

divisors\_sum = sum(i for i in range(1, num) if num % i == 0)

    if divisors\_sum == num:

        return "Perfect"

    elif divisors\_sum < num:

        return "Deficient"

    else:

        return "Abundant"

def classify\_numbers\_in\_range(start, end):

classification = {}

    for num in range(start, end + 1):

        category = classify\_number(num)

        classification[num] = category

return classification

# Example usage for numbers in the range 1 to 30

start\_range = 1

end\_range = 30

result = classify\_numbers\_in\_range(start\_range, end\_range)

for num, category in result.items():

    print(f"{num} is a {category} number.")

1. Write a program that plays a “hi-low” number guessing game. The program starts by generating a random number hidden from the user that is between 0 and 100. The user then attempts to guess the number, getting hints as to which direction (bigger or smaller, higher or lower) to go on the next guess. The game can end in one of two ways:

-The user can correctly guess the number.

-The user can quit playing by entering a number out of the range of 0– 100.

import random

def hi\_low\_game():

    # Generate a random number between 0 and 100

    secret\_number = random.randint(0, 100)

    print("Welcome to the Hi-Low Number Guessing Game!")

    print("Try to guess the hidden number between 0 and 100.")

print("Enter a number outside this range to quit.")

    while True:

        try:

            # Get user's guess

            user\_guess = int(input("Enter your guess: "))

            # Check if the guess is within the valid range

            if 0 <= user\_guess <= 100:

                # Provide hints and check if the guess is correct

                if user\_guess < secret\_number:

                    print("Go higher!")

                elif user\_guess > secret\_number:

                    print("Go lower!")

                else:

                    print("Congratulations! You guessed the correct number.")

                    break

            else:

                print("Quitting the game. The correct number was", secret\_number)

                break

        except ValueError:

            print("Invalid input. Please enter a valid number.")

# Start the game

hi\_low\_game()

1. Write a program to generate the hailstone sequence which is formed by applying the formula below to the initial number and then repeatedly to each number generated by the formula. The result is a sequence of integers ending at 1. For example, if you start with 5, you get the following sequence: 5, 16, 8, 4, 2, 1. The hailstone formula is as follows:

-If the number is even, divide it by 2.

-If the number is odd, multiply by 3 and add 1.

-When the number reaches 1, quit.

def hailstone\_sequence(n):

    sequence = [n]

    while n != 1:

        if n % 2 == 0:

            n //= 2

        else:

            n = 3 \* n + 1

        sequence.append(n)

return sequence

# Example usage

initial\_number = 9

result\_sequence = hailstone\_sequence(initial\_number)

print(f"Hailstone sequence for {initial\_number}: {result\_sequence}")