ASSIGNMENT 2.1

TASK-1

CODE:

A green screen with red text

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

OUTPUT:

A white background with black text

AI-generated content may be incorrect.

EXPLANATION:

1. **def calculate\_stats(numbers):**: This line defines a function named calculate\_stats that accepts one argument, which is expected to be a list of numbers, and is referred to as numbers within the function.
2. **""" ... """**: This is a docstring, which explains what the function does, its arguments (Args), and what it returns (Returns). This is good practice for documenting your code.
3. **if not numbers:**: This checks if the input list numbers is empty.
4. **return None, None, None**: If the list is empty, the function returns None for the mean, minimum, and maximum, as these values cannot be calculated for an empty list.
5. **mean = sum(numbers) / len(numbers)**: If the list is not empty, this line calculates the mean. sum(numbers) calculates the sum of all elements in the list, and len(numbers) gets the number of elements in the list. The sum is then divided by the count to get the mean.
6. **minimum = min(numbers)**: This line uses the built-in Python function min() to find the smallest value in the list.
7. **maximum = max(numbers)**: This line uses the built-in Python function max() to find the largest value in the list.
8. **return mean, minimum, maximum**: Finally, the function returns the calculated mean, minimum, and maximum values as a tuple.

The code also includes an example of how to use the function:

1. **my\_list = [10, 20, 30, 40, 50]**: This creates a sample list of numbers.
2. **mean\_value, min\_value, max\_value = calculate\_stats(my\_list)**: This calls the calculate\_stats function with my\_list and unpacks the returned tuple into three variables: mean\_value, min\_value, and max\_value.
3. **print(...)**: These lines print the original list and the calculated mean, minimum, and maximum values.
4. **empty\_list = []**: This creates an empty list to demonstrate how the function handles this case.
5. **mean\_empty, min\_empty, max\_empty = calculate\_stats(empty\_list)**: This calls the function with the empty list.
6. **print(...)**: These lines print the empty list and the None values returned by the function

TASK-2

CODE:

A screenshot of a computer

AI-generated content may be incorrect.

OUTPUT:



EXPLANATION:

Certainly! This Python function is\_armstrong\_number checks if a given number is an Armstrong number. Here's a breakdown of how it works:

1. **Convert to String**: The number is first converted into a string (num\_str). This makes it easy to iterate through each digit and determine the total number of digits.
2. **Get Number of Digits**: The length of the string (num\_digits) gives the count of digits in the original number.
3. **Initialize Sum**: A variable sum\_of\_powers is initialized to 0. This variable will store the sum of each digit raised to the power of the total number of digits.
4. **Iterate and Calculate Sum**: The code then loops through each character (digit) in the string. For each digit:
   * It's converted back to an integer (int(digit)).
   * It's raised to the power of num\_digits.
   * The result is added to sum\_of\_powers.
5. **Check for Armstrong Number**: Finally, the function compares sum\_of\_powers with the original number. If they are equal, it means the number is an Armstrong number, and the function returns True. Otherwise, it returns False.

In simple terms, the function takes a number, calculates the sum of its digits raised to the power of the total number of digits, and then checks if this sum is the same as the original number.

0 / 2000

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](http://g.co/legal/generative-code)

COPILOT CODE:

A screenshot of a computer program

AI-generated content may be incorrect.

OUTPUT:



TASK-3

CODE:

A green screen with text

AI-generated content may be incorrect.

OUTPUT:



EXPLANATION:

Sure, let's go through the is\_prime function line by line as requested:

* **Line 1 (if n <= 1:)**: This line checks if the input number n is less than or equal to 1. By definition, prime numbers are natural numbers greater than 1. So, if the number is 1 or less, it's not prime, and the function immediately returns False.
* **Line 2 (for i in range(2, int(math.sqrt(n)) + 1):)**: This line starts a loop to check for divisors of n. It iterates through numbers i starting from 2 up to the integer part of the square root of n. We only need to check up to the square root because if a number n has a divisor greater than its square root, it must also have a divisor smaller than its square root. The import math is used to access the sqrt function.
* **Line 3 (if n % i == 0:)**: Inside the loop, this line checks if n is perfectly divisible by the current number i. The modulo operator (%) gives the remainder of the division. If the remainder is 0, it means n is divisible by i, and therefore n is not a prime number. In this case, the function returns False.
* **Line 4 (return True)**: If the loop completes without finding any divisors (i.e., the if n % i == 0: condition was never met), it means the number n has no divisors other than 1 and itself. Therefore, the number is prime, and the function returns True.

TASK-4

CODE:

A screenshot of a computer code

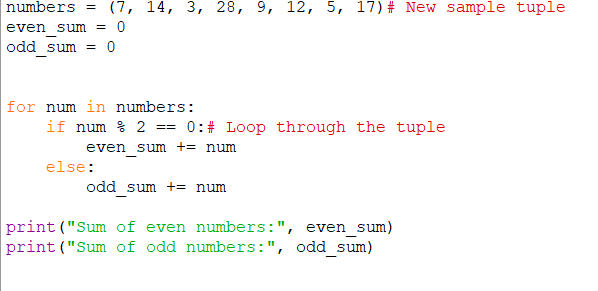
AI-generated content may be incorrect.

OUTPUT:



TASK-5

CODE:



OUTPUT:

A close up of a number

AI-generated content may be incorrect.