

AI-ASSISTED CODING ASSIGNMENT 3.2

SHRUTHI ADDAGUDI | 2303A52133 | BATCH 37

Task Description-1

Progressive Prompting for Calculator Design: Ask the AI to design a simple calculator program by initially providing only the function name. Gradually enhance the prompt by adding comments and usage examples.

Prompt used:

design a simple calculator program by initially providing only the function name

code explanation:

Here, only the function name `simple_calculator()` is defined with a `pass` statement. Since no instructions or logic were provided, the AI generated an empty function. This shows that minimal prompts result in incomplete code.

The screenshot shows a code editor on the left and an AI interface on the right. The code editor has a dark theme and displays the following Python code:

```
[2] 0s def simple_calculator():
    pass
```

The AI interface is titled "Gemini" and contains the following text:

Okay, let's start by defining the function name for your simple calculator program.
The `simple_calculator` function has been successfully defined. It currently does nothing because it's empty. How would you like to proceed with implementing its functionality?

Prompt used:

design a calculator function that performs addition, subtraction, multiplication, and division

code explanation:

Here, the function accepts two numbers and an operator to perform arithmetic operations. Conditional statements are used to handle addition, subtraction, multiplication, and division. Error handling is added for division by zero and invalid operators due to clearer prompt instructions.

The screenshot shows a code editor on the left and an AI interface on the right. The code editor contains a Python script named `simple_calculator.py` with the following content:

```
[9] 0s
def simple_calculator(num1, num2, operator):
    if operator == '+':
        return num1 + num2
    elif operator == '-':
        return num1 - num2
    elif operator == '*':
        return num1 * num2
    elif operator == '/':
        if num2 == 0:
            return "Error: Division by zero is not allowed."
        return num1 / num2
    else:
        return "Error: Invalid operator. Please use '+', '-', '*', or '/'."

```

The AI interface on the right is titled "Gemini". It has a sidebar with "RAM" and "Disk" indicators. The main area shows a history of interactions:

- S** design a calculator function that performs addition, subtraction, multiplication, and division
- G** I will now implement the `simple_calculator` function to perform addition, subtraction, multiplication, and division, along with error handling.
- The `simple_calculator` function has been successfully defined with addition, subtraction, multiplication, and division capabilities. It also includes error handling for division by zero and invalid operators. You can now use it to perform calculations!

Prompt used:

Create a calculator function that performs addition, subtraction, multiplication, and division with function names

Code explanation:

In this stage, the calculator is divided into separate functions for add, subtract, multiply, and divide operations. This modular approach improves code readability and reusability. The AI generated cleaner and well-structured code based on refined prompting.

The screenshot shows a code editor on the left and an AI interface on the right. The code editor contains four separate functions: `add`, `subtract`, `multiply`, and `divide`.

```
[9] 0s
def add(num1, num2):
    return num1 + num2

def subtract(num1, num2):
    return num1 - num2

def multiply(num1, num2):
    return num1 * num2

def divide(num1, num2):
    if num2 == 0:
        return "Error: Division by zero is not allowed."
    return num1 / num2

```

The AI interface on the right is titled "Gemini". It has a sidebar with "RAM" and "Disk" indicators. The main area shows a history of interactions:

- S** Create a calculator function that performs addition, subtraction, multiplication, and division with function names
- G** Certainly! I'll create separate functions for addition, subtraction, multiplication, and division. Each function will take two numbers as input, and the division function will include error handling for division by zero.
- The individual calculator functions (`add`, `subtract`, `multiply`, and `divide`) have been successfully defined. You can now use them to perform calculations!

Below the history, there are several suggestions in a rounded rectangle:

- Combine the add, subtract, multiply, and divide functions into a single calculator function
- Explain how to use the divide function with error handling
- Test the add and subtract functions

At the bottom, there is a question: "What can I help you build?"

Prompt used:

Create a calculator function that performs addition, subtraction, multiplication, and division with function names and give examples for calculator performing for all functions

Output:

$10 + 5 = 15$

$10 - 5 = 5$

$10 * 5 = 50$

10/5 = 2

Code explanation:

Code gives us the execution and testing of all calculator functions using sample inputs. The outputs verify correct calculations and proper error handling. Including usage examples in the prompt helped the AI generate accurate test cases.

```
# Example for addition
print("10 + 5 = " + str(add(10, 5)))
10 + 5 = 15

# Example for subtraction
print("10 - 5 = " + str(subtract(10, 5)))
10 - 5 = 5

# Example for multiplication
print("10 * 5 = " + str(multiply(10, 5)))
10 * 5 = 50

# Example for division
print("10 / 5 = " + str(divide(10, 5)))

# Example for division by zero (error handling)
print("10 / 0 = " + str(divide(10, 0)))
10 / 0 = Error: Division by zero is not allowed.
```

The screenshot shows a code editor with several print statements demonstrating basic arithmetic operations (addition, subtraction, multiplication, division) and error handling for division by zero. The output of the code is displayed below each print statement, showing the results or an error message.

Gemini Response:

S Create a calculator function that performs addition, subtraction, multiplication, and division with function names and give examples for calculator performing for all functions

◆ Certainly! Here are examples demonstrating how to use each of the individual calculator functions: `add`, `subtract`, `multiply`, and `divide`.

The examples have executed successfully, demonstrating each function:

- `add(10, 5)` returns 15.
- `subtract(10, 5)` returns 5.
- `multiply(10, 5)` returns 50.

Refactor the 'simple_calculator' function to return specific error messages
Explain the error handling in the 'divide' function
Combine the individual functions into a single calculator class

What can I help you build?

+ Gemini 2.5 Flash >

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

Task Description-2

- Refining Prompts for Sorting Logic: Start with a vague prompt for sorting student marks, then refine it to clearly specify sorting order and constraints

prompt used :

Sort student marks

```
# Sample list of student marks
student_marks = [85, 92, 78, 65, 95, 88, 70, 90]

# Sort the marks in ascending order
sorted_marks_ascending = sorted(student_marks)
print(f"Marks sorted in ascending order: {sorted_marks_ascending}")

# Sort the marks in descending order
sorted_marks_descending = sorted(student_marks, reverse=True)
print(f"Marks sorted in descending order: {sorted_marks_descending}")

... Marks sorted in ascending order: [65, 70, 78, 85, 88, 90, 92, 95]
Marks sorted in descending order: [95, 92, 90, 88, 85, 78, 70, 65]
```

The screenshot shows a code editor with Python code that defines a list of student marks and uses the `sorted()` function to sort them both in ascending and descending order. The output of the code is displayed below each print statement, showing the sorted lists.

Gemini Response:

S Sort student marks

◆ Sure, I can help you sort student marks. To start, I'll show you how to sort a simple list of numerical marks in ascending order.

The student marks have been successfully sorted and displayed:

- Ascending Order: [65, 70, 78, 85, 88, 90, 92, 95]
- Descending Order: [95, 92, 90, 88, 85, 78, 70, 65]

This demonstrates how to use the `sorted()` function in Python to sort a list of numbers.

Calculate the average of the sorted student marks
Find the median of the sorted student marks

Code explanation:

The list `student_marks` stores the marks of different students. The `sorted()` function is used first to arrange the marks in ascending order and then in

descending order by setting reverse=True. This demonstrates how clear instructions help in sorting data accurately in different orders.

Prompt used:

Write a Python function to sort student marks in descending order without using built-in sorting functions.

The screenshot shows a code editor on the left and a Gemini AI interface on the right. The code editor contains Python code for a function named `manual_sort_descending`. The function iterates through the array, finds the maximum element in the remaining unsorted portion, and swaps it with the first element of the unsorted portion. A sample list of student marks is provided, and the sorted list is printed. The Gemini interface has a prompt asking for a Python function to sort marks in descending order without built-in functions. It also shows a generated response explaining the selection sort algorithm and providing the sorted list [95, 92, 90, 88, 85, 78, 70, 65].

```
def manual_sort_descending(marks):
    n = len(marks)
    # Traverse through all marks elements
    for i in range(n):
        # Find the maximum element in remaining unsorted array
        max_idx = i
        for j in range(i + 1, n):
            if marks[j] > marks[max_idx]:
                max_idx = j
        # Swap the found maximum element with the first element
        marks[i], marks[max_idx] = marks[max_idx], marks[i]
    return marks

# Sample list of student marks
student_marks = [85, 92, 78, 65, 95, 88, 70, 90]

# Sort the marks in descending order using the custom function
sorted_marks_manual = manual_sort_descending(list(student_marks)) # Use list() to create a copy to avoid
print("Marks sorted in descending order : {sorted_marks_manual}")
```

Gemini

S Write a Python function to sort student marks in descending order without using built-in sorting functions.

I will create a Python function named `manual_sort_descending` that sorts a list of student marks in descending order using the selection sort algorithm, without relying on built-in sorting functions. Then, I'll provide an example to demonstrate its use.

The custom `manual_sort_descending` function has successfully sorted the student marks without using built-in sorting functions. The output shows:

Marks sorted in descending order (manual sort): [95, 92, 90, 88, 85, 78, 70, 65]

This confirms the function works as expected.

Explain the selection sort algorithm used
Calculate the average of the sorted marks
Implement the custom sort function for ascending order

What can I help you build?

Code explanation:

The function implements the **Selection Sort** algorithm to arrange student marks in descending order without using built-in sorting methods. It works by iterating through the list, finding the maximum element in the unsorted portion during each pass. This maximum element is then swapped into its correct position, and the process repeats until the entire list is sorted from largest to smallest.

Task Description-3

Few-Shot Prompting for Prime Number Validation: Provide multiple input-output examples for a function that checks whether a number is prime. Observe how few-shot

prompting improves correctness.

Prompt used:

```

[17] ✓ 0s
def is_prime(num):
    if num < 2:
        return "Not Prime"
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return "Not Prime"
    return "Prime"

# Example calls
print(f"Input: 2 \u2192 Output: {is_prime(2)}")
print(f"Input: 4 \u2192 Output: {is_prime(4)}")
print(f"Input: 1 \u2192 Output: {is_prime(1)}")
print(f"Input: 17 \u2192 Output: {is_prime(17)}")

Input: 2 → Output: Prime
Input: 4 → Output: Not Prime
Input: 1 → Output: Not Prime
Input: 17 → Output: Prime

```

Gemini

- S Write a function to check whether a number is prime. Examples: Input: 2 → Output: Prime Input: 4 → Output: Not Prime Input: 1 → Output: Not Prime Input: 17 → Output: Prime
- ◆ Certainly! I'll create a Python function `is_prime` that checks whether a given number is prime, and then demonstrate it with the examples you provided.

The `is_prime` function has been successfully executed, and the results match the expected outputs:

- Input 2 correctly returns `Prime`.
- Input 4 correctly returns `Not Prime`.
- Input 1 correctly returns `Not Prime`.
- Input 17 correctly returns `Prime`.

[Explain the 'is_prime' function code](#)
[Test the 'is_prime' function with a list of numbers](#)
[How to find the average of the sorted student marks](#)

Code explanation:

This function determines if a number is prime. It first checks if the number is less than 2, returning 'Not Prime' if true. Otherwise, it efficiently iterates from 2 up to the square root of the number. If any number in this range divides the input evenly, it's 'Not Prime'; otherwise, the number is declared 'Prime'.

Task Description-4

Guided UI Design for Student Grading System

Prompt used :

Create a user interface for a student grading system that calculates total marks, percentage, and grade based on user input.

```

Marks sorted in descending order : [95, 92, 90, 88, 85, 78, 70, 65]
[18] ✓ 16s
student_marks_input = []
while True:
    mark_input = input("Enter student mark (or 'done' to finish): ")
    if mark_input.lower() == 'done':
        break
    try:
        mark = float(mark_input)
        if mark < 0 or mark > 100: # Optional: Add validation for mark range
            print("Mark should be between 0 and 100. Please try again.")
        else:
            student_marks_input.append(mark)
    except ValueError:
        print("Invalid input. Please enter a number or 'done'.")
if student_marks_input:
    total_marks_input = sum(student_marks_input)
    print(f"Entered marks: {student_marks_input}")
    print(f"Total marks: {total_marks_input}")
else:
    print("No marks were entered.")

...
Enter student mark (or 'done' to finish): 56
Enter student mark (or 'done' to finish): 89
Enter student mark (or 'done' to finish): done
Entered marks: [56.0, 89.0]
Total marks: 145.0

```

Gemini

- ◆ Sure, here's a Python program that will allow you to input student marks one by one and then display the total sum of those marks.

The program successfully accepted the student marks you entered (56.0, 89.0) and correctly calculated their total, which is 145.0.

[Calculate the average of the sorted student marks](#)
[Display the median of the student marks](#)
[Explain the logic of the `is_prime` function](#)
[What can I help you build?](#)

[Gemini 2.5 Flash](#)

Gemini can make mistakes - so double-check it and use code with caution. [Learn more](#)

Code explanation:

This program initializes an empty list and enters a loop to continuously accept student marks as input. It validates each mark, ensuring it's a number between 0 and 100, and adds valid entries to the list, while gracefully handling non-numeric input. The loop continues until the user types 'done', at which point it

exits. Finally, if any marks were entered, the program calculates and displays both the list of entered marks and their total sum.

Prompt used:

Create a Python program that accepts marks of 5 subjects from the user, calculates total marks, percentage, and displays grade based on percentage.

The screenshot shows a terminal window with two panes. The left pane contains the Python code for calculating marks and grades. The right pane is a Gemini AI interface providing feedback and analysis of the code.

Code Content:

```
[19] 27s
marks = []
num_subjects = 5
max_marks_per_subject = 100

print(f"Please enter marks for {num_subjects} subjects (out of {max_marks_per_subject} each):")

for i in range(num_subjects):
    while True:
        try:
            mark_input = input(f"Enter mark for subject {i + 1}: ")
            mark = float(mark_input)
            if 0 <= mark <= max_marks_per_subject:
                marks.append(mark)
                break
            else:
                print("Invalid mark. Please enter a mark between 0 and", max_marks_per_subject)
        except ValueError:
            print("Invalid input. Please enter a numeric value.")

# Calculate total marks
total_marks = sum(marks)

# Calculate percentage
max_possible_total = num_subjects * max_marks_per_subject
percentage = (total_marks / max_possible_total) * 100

# Determine grade based on percentage
if percentage >= 90:
    grade = 'A'
elif percentage >= 70:
    grade = 'B+'
elif percentage >= 60:
    grade = 'B'
elif percentage >= 50:
    grade = 'C'
elif percentage >= 40:
    grade = 'D'
else:
    grade = 'F'

# Display results
print("\n--- Results ---")
print(f"Marks entered: {marks}")
print(f"Total Marks: {total_marks:.2f} / {max_possible_total}")
print(f"Percentage: {percentage:.2f}%")
print(f"Grade: {grade}")

... Please enter marks for 5 subjects (out of 100 each):
Enter mark for subject 1: 56
Enter mark for subject 2: 78
Enter mark for subject 3: 98
Enter mark for subject 4: 89
Enter mark for subject 5: 88

--- Results ---
Marks entered: [56.0, 78.0, 98.0, 89.0, 88.0]
Total Marks: 409.00 / 500
Percentage: 81.80%
Grade: A
```

Gemini Analysis:

- It prompted you to enter marks for 5 subjects.
- The marks entered were [56.0, 78.0, 98.0, 89.0, 88.0].
- The Total Marks calculated are 409.00 out of a possible 500.

The program has successfully executed and processed the student marks:

- The Percentage is 81.80%

What can I help you build?

+ Gemini 2.5 Flash ➔

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

Code explanation:

This program prompts the user to enter marks for five subjects, validating each input to ensure it's a number between 0 and 100. It then calculates the total

marks and the overall percentage achieved by the student. Finally, based on a predefined grading scale, it assigns and displays a corresponding letter grade for the student's performance.

Task Description-5

Analysing Prompt Specificity in Unit Conversion Functions: Improving a Unit Conversion Function (Kilometres to Miles and Miles to Kilometres) Using Clear Instructions.

Prompt used :

Create a Python program with two functions:

1. Convert kilometers to miles
2. Convert miles to kilometers

Use accurate conversion values and display the results clearly.

The screenshot shows a code editor on the left and a Gemini AI interface on the right. The code editor contains Python code for unit conversion functions and example usage. The Gemini interface shows the user prompt, AI response, and the resulting output.

```
Start coding or generate with AI.

def kilometers_to_miles(kilometers):
    """Converts kilometers to miles."""
    # 1 kilometer = 0.621371 miles
    miles = kilometers * 0.621371
    return miles

def miles_to_kilometers(miles):
    """Converts miles to kilometers."""
    # 1 mile = 1.60934 kilometers
    kilometers = miles * 1.60934
    return kilometers

# Example usage:
# Convert kilometers to miles
kms = 10
miles_result = kilometers_to_miles(kms)
print(f"{kms} kilometers is equal to {miles_result:.2f} miles.")

kms = 100
miles_result = kilometers_to_miles(kms)
print(f"{kms} kilometers is equal to {miles_result:.2f} miles.")

# Convert miles to kilometers
mil = 20
kilometers_result = miles_to_kilometers(mil)
print(f"{mil} miles is equal to {kilometers_result:.2f} kilometers.

mil = 60
kilometers_result = miles_to_kilometers(mil)
print(f"{mil} miles is equal to {kilometers_result:.2f} kilometers.")
```

Gemini

S Create a Python program with two functions: 1. Convert kilometers to miles. 2. Convert miles to kilometers. Use accurate conversion values and display the results clearly.

◆ Sure, I can create a Python program with two functions to convert between kilometers and miles using accurate conversion values. I will also include examples to clearly display the results.

The conversion functions have executed successfully!

Here are the results:

- 10 kilometers is equal to 6.21 miles.
- 100 kilometers is equal to 62.14 miles.
- 20 miles is equal to 32.19 kilometers.
- 60 miles is equal to 96.56 kilometers.

The unit conversion program has been successfully created and executed, and the results were displayed. Is there anything else you would like to do or a new task you'd like to work on?

What can I help you build?

+ Gemini 2.5 Flash ▶

Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

Code explanation:

The Python program defines two functions: kilometers_to_miles and miles_to_kilometers. These functions facilitate unit conversion between kilometers and miles using precise conversion factors. The code then demonstrates their usage with various input values, clearly printing the converted results to the console, formatted to two decimal places for readability.