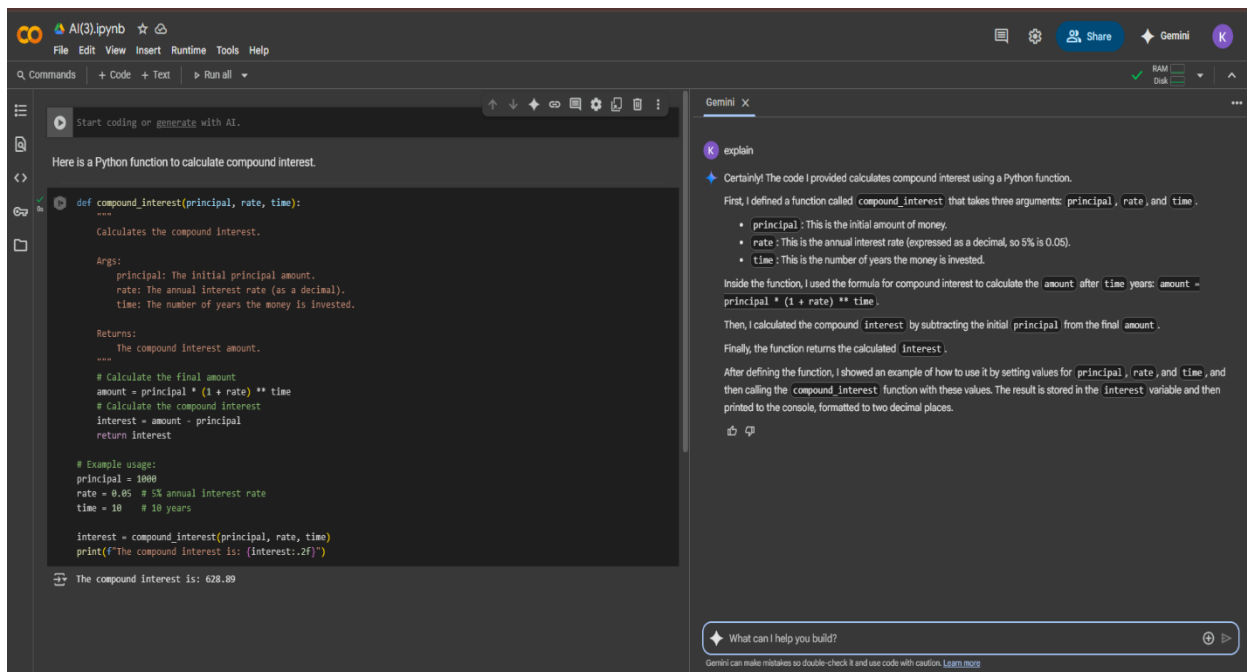
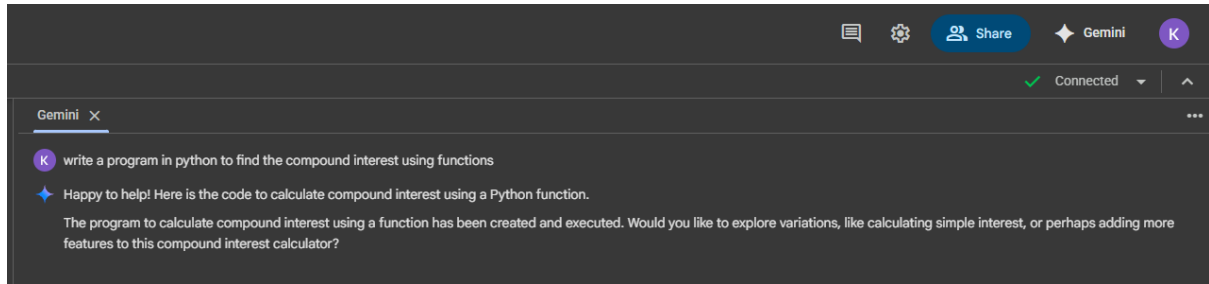


Assignment 3.2

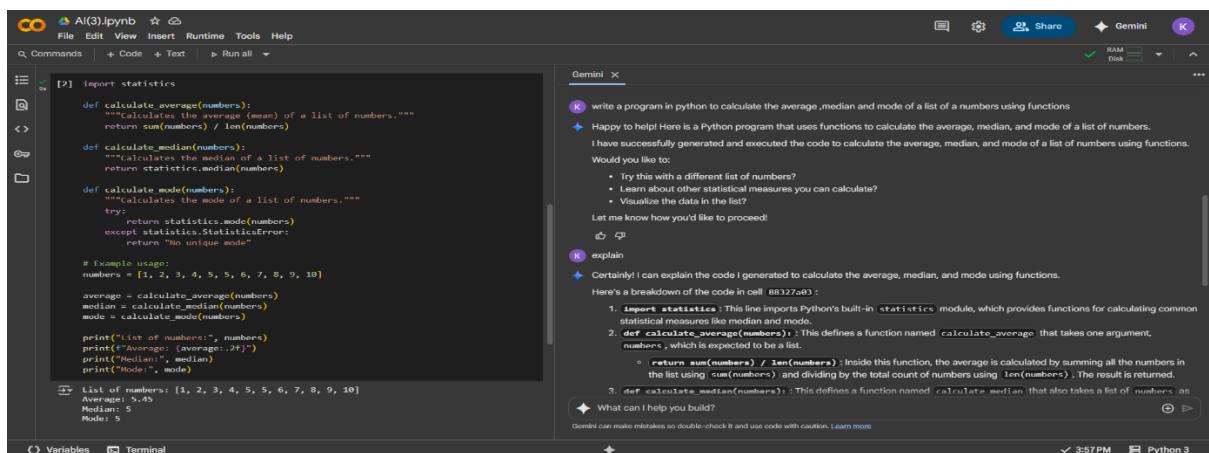
Name: K. Vinod

2403A52132

Task 1:



Task 2:



Explanation:

1. `import statistics`: This line imports Python's built-in `statistics` module, which provides functions for calculating common statistical measures like median and mode.

2. `def calculate_average(numbers):`: This defines a function named `calculate_average` that takes one argument, `numbers`, which is expected to be a list.

- `return sum(numbers) / len(numbers)`: Inside this function, the average is calculated by summing all the numbers in the list using `sum(numbers)` and dividing by the total count of numbers using `len(numbers)`. The result is returned.

3. `def calculate_median(numbers):`: This defines a function named `calculate_median` that also takes a list of `numbers` as input.

- `return statistics.median(numbers)`: This line uses the `median()` function from the imported `statistics` module to calculate the median of the list and returns the result.

4. `def calculate_mode(numbers):`: This defines a function named `calculate_mode` that takes a list of `numbers`.

- `try:` This starts a `try` block to handle potential errors. The `statistics.mode()` function can raise a `StatisticsError` if there is no unique mode (e.g., a list with multiple numbers appearing with the same highest frequency).
- `mode = statistics.mode(numbers)`: This line attempts to calculate the mode using the `mode()` function from the `statistics` module and assigns it to the `mode` variable.
- `return mode`: If the mode is successfully calculated, it is returned.
- `except statistics.StatisticsError:` If a `StatisticsError` occurs (meaning there's no unique mode), the code inside this `except` block is executed.
- `return "No unique mode"`: In the case of no unique mode, the function returns the string "No unique mode".

5. `# Example usage:`: This is a comment indicating the start of the example usage of the defined functions.

6. `numbers = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]`: A sample list of numbers is created and assigned to the `numbers` variable.

7. `average = calculate_average(numbers)`: The `calculate_average` function is called with the `numbers` list, and the returned average is stored in the `average` variable.

8. `median = calculate_median(numbers)`: The `calculate_median` function is called with the `numbers` list, and the returned median is stored in the `median` variable.

9. `mode = calculate_mode(numbers)`: The `calculate_mode` function is called with the `numbers` list, and the returned mode (either the value or the "No unique mode" string) is stored in the `mode` variable.

10. `print(...)`: The final `print` statements display the original list and the calculated average, median, and mode with descriptive labels. The average is formatted to two decimal places using an f-string (`f"{average:.2f}"`).

In summary, the code defines three functions to perform specific statistical calculations and then demonstrates how to use these functions with a sample list of numbers, handling the case where a unique mode might not exist.

Task 3:

The screenshot displays the Google AI Studio interface. On the left, a code editor shows a Python program for converting decimal numbers to binary. The program defines a function `decimal_to_binary` that takes a decimal number and returns its binary representation using `bin()`. It then uses this function to convert several numbers (10, 25, 0, 100, 255, 1024) and prints the results. The output shows the decimal numbers and their binary representations. On the right, the Gemini chat interface shows a conversation where the user asks for a program to convert decimal values to binary, and the AI responds with a similar program and examples.

Explanation:

```
...
K Explanation
➦ Absolutely! I can explain the code that converts decimal numbers to binary.
Here's a breakdown of the code in cell b4cbc4d9:

1. def decimal_to_binary(decimal_num): : This defines a function named decimal_to_binary that takes one argument, decimal_num, which is expected to be a decimal integer.
    ◦ """Converts a decimal integer to its binary representation.""" : This is a docstring, which explains what the function does.
    ◦ return bin(decimal_num) : This is the core of the function. Python has a built-in function called bin() that directly converts an integer to its binary string representation. The result is returned by the function. The binary string returned by bin() is prefixed with 0b to indicate that it is a binary number.

2. # Example usage with multiple examples: : This is a comment indicating the start of the examples.

3. decimal_numbers = [10, 25, 0, 100, 255, 1024] : A list of decimal integers is created to demonstrate the conversion with different values.

4. print("Converting decimal numbers to binary:") : This line prints a header to the output.

5. for num in decimal_numbers: : This starts a for loop that iterates through each number in the decimal_numbers list.
    ◦ binary_representation = decimal_to_binary(num) : Inside the loop, the decimal_to_binary function is called for each num in the list, and the returned binary string is stored in the binary_representation variable.
    ◦ print(f"The decimal number {num} is {binary_representation} in binary.") : This line prints the original decimal number and its binary representation using an f-string for formatted output.

6. # You can also directly use the bin() function: : This is a comment showing that you don't necessarily need a separate function for this simple conversion if you just need to use it once or twice.

7. print("\nUsing the built-in bin() function directly:") : This prints another header for the direct usage examples.

8. print(f"The decimal number 7 is {bin(7)} in binary.") : This line directly uses the bin() function to convert the integer 7 to binary and prints the result.

9. print(f"The decimal number 32 is {bin(32)} in binary.") : Similarly, this line directly uses the bin() function to convert the integer 32 to binary and prints the result.

In essence, the code defines a simple function that wraps the built-in bin() function for clarity and then provides multiple examples demonstrating how to use this function and how to use the bin() function directly to convert decimal integers to their binary string representations.

⚡ What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more
🕒 4:13 PM 📄 Python 3
```

Task 4:

Step 1: Design the user interface

Let's start by creating a basic user interface using Tkinter to get customer details and requirements. This initial code will set up the main window and some input fields.

```
def generate_bill():
    """Prompts the user for item names and quantities, calculates total, and generates a simple bill."""
    items = {}
    # Define a dictionary with item prices
    item_prices = {
        "idly": 10,
        "vada": 12,
        "coffee": 15,
        "pizza": 100
    }

    print("Enter item names and quantities (type 'done' when finished):")
    print("Available items and prices:")
    for item, price in item_prices.items():
        print(f"{item}: ${price:.2f}")
    print("-" * 20)

    total_cost = 0

    while True:
        item_name = input("Enter item name: ").strip().lower()
        if item_name == 'done':
            break
        if item_name not in item_prices:
            print(f"Sorry, '{item_name}' is not on the menu.")
            continue

        try:
            quantity = int(input(f"Enter quantity for {item_name}: ").strip())
            if quantity <= 0:
                print("Quantity must be a positive number.")
                continue

            items[item_name] = items.get(item_name, 0) + quantity
            total_cost += item_prices[item_name] * quantity

    # Print the bill
    print("\n--- Your Bill ---")
    if not items:
        print("No items added to the bill.")
    else:
        for item, quantity in items.items():
            price = item_prices[item]
            print(f"{item.capitalize():15} {quantity} x ${price:.2f} = ${quantity * price:.2f}")
        print("-" * 20)
        print(f"Total: ${total_cost:.2f}")
        print("-" * 20)

    # Run the bill generation function
    generate_bill()
```

Gemini

create a user interface for an hotel to generate bill based on customer requirements

I've added the logic to the `generate_bill` function in the Tkinter application to calculate and display the hotel bill based on the entered details. Please note that this GUI will only run in a local Python environment with Tkinter installed, as Colab does not directly support running graphical user interfaces in the notebook cells.

I have added the functionality to generate the bill within the Tkinter application. As mentioned before, you would need to run this code in a local Python environment to see the GUI.

Here are some possible next steps you could take with this code:

- **Enhance the Tkinter GUI:** Add more input fields for things like check-in/check-out dates, number of guests, or a list of available services/menu items for the user to select from.
- **Implement persistent storage:** Save the billing information to a file (like a CSV or a simple text file) so you can keep a record of transactions.
- **Add more complex calculations:** Include taxes, discounts, or different pricing tiers based on room occupancy or peak seasons.
- **Integrate with a database:** For a more robust system, you could store the data in a database like SQLite, PostgreSQL, or MySQL.
- **Explore different GUI libraries:** If you're interested in creating more visually appealing interfaces, you could try libraries like PyQt or Kivy.

send me a code

Could you please clarify what kind of code you're looking for? For example, are you looking for a specific function, a complete program, or something related to the previous hotel billing system?

please input item name like idly vada coffee pizza and its quantities from the user and generate the bill

Okay, I can help you with that. Here's a Python code snippet that prompts the user to enter item names and quantities

What can I help you build?

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

```
total_cost = 0

while True:
    item_name = input("Enter item name: ").strip().lower()
    if item_name == 'done':
        break
    if item_name not in item_prices:
        print(f"Sorry, '{item_name}' is not on the menu.")
        continue

    try:
        quantity = int(input(f"Enter quantity for {item_name}: ").strip())
        if quantity <= 0:
            print("Quantity must be a positive number.")
            continue

        if item_name in items:
            items[item_name] += quantity
        else:
            items[item_name] = quantity
        total_cost += item_prices[item_name] * quantity

    except ValueError:
        print("Invalid quantity. Please enter a number.")

print("\n--- Your Bill ---")
if not items:
    print("No items added to the bill.")
else:
    for item, quantity in items.items():
        price = item_prices[item]
        print(f"{item.capitalize():15} {quantity} x ${price:.2f} = ${quantity * price:.2f}")
    print("-" * 20)
    print(f"Total: ${total_cost:.2f}")
    print("-" * 20)

# Run the bill generation function
generate_bill()
```

Gemini

send me a code

Could you please clarify what kind of code you're looking for? For example, are you looking for a specific function, a complete program, or something related to the previous hotel billing system?

please input item name like idly vada coffee pizza and its quantities from the user and generate the bill

Okay, I can help you with that. Here's a Python code snippet that prompts the user to enter item names and quantities and then calculates and displays a simple bill.

I have provided the code to input item names and quantities and generate a basic bill.

Here are some ways you could enhance this billing program:

- **Add prices for items:** Modify the code to include a dictionary or another data structure to store the prices of different items (like idly, vada, coffee, pizza).
- **Calculate the total cost:** Based on the quantities and prices, calculate the total cost of the items and display it on the bill.
- **Include tax:** Add a tax calculation to the total bill.
- **Implement discounts:** Allow for discounts based on certain criteria (e.g., a percentage discount for a minimum total).
- **Save the bill:** Store the generated bill in a file for record-keeping.
- **Create a simple menu:** Display a list of available items and their prices to the user before they start entering their order.

and update the code

Okay, I will update the bill generation code to include item prices and calculate the total cost.

I have updated the code to include item prices and calculate the total bill.

What can I help you build?

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

```
price = item_prices[item]
print(f"{item.capitalize():15} {quantity} x ${price:.2f} = ${quantity * price:.2f}")
print("-" * 20)
print(f"Total: ${total_cost:.2f}")
print("-" * 20)

# Run the bill generation function
generate_bill()
```

Enter item names and quantities (type 'done' when finished):

Available items and prices:

- idly: \$10.00
- vada: \$12.00
- coffee: \$15.00
- pizza: \$100.00

Enter item name: idly

Enter quantity for idly: 4

Enter item name: vada

Enter quantity for vada: 2

Enter item name: done

--- Your Bill ---

Idly: 4 x \$10.00 = \$40.00

Vada: 2 x \$12.00 = \$24.00

Total: \$64.00

Gemini

- **Implement discounts:** Allow for discounts based on certain criteria (e.g., a percentage discount for a minimum total).
- **Save the bill:** Store the generated bill in a file for record-keeping.
- **Create a simple menu:** Display a list of available items and their prices to the user before they start entering their order.

and update the code

Okay, I will update the bill generation code to include item prices and calculate the total cost.

I have updated the code to include item prices and calculate the total bill.

Here are some further improvements or related tasks you could consider:

- **Add error handling for invalid input:** The current code has basic error handling for non-numeric quantity input, but you could add more robust handling for other potential issues.
- **Implement discounts or taxes:** Extend the calculation to include discounts or add tax to the total bill.
- **Save the bill to a file:** Write the generated bill details to a text file or CSV file for record-keeping.
- **Create a more interactive menu:** Instead of just printing the menu, you could present it in a more user-friendly way, perhaps with numbers corresponding to items for easier selection.
- **Add more items and prices:** Expand the `item_prices` dictionary to include a wider variety of items.
- **Develop a simple GUI:** If you want a visual interface, you could use a library like Tkinter (as in the previous example) to create a graphical billing application.

What can I help you build?

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

Explanation:

```
def generate_bill():
    """Prompts the user for item names and quantities, calculates total, and generates a simple bill."""
    items = {}
    # Define a dictionary with item prices
    item_prices = {
        "soda": 10,
        "sand": 15,
        "coffee": 15,
        "pizza": 100
    }

    print("Enter item names and quantities (type 'done' when finished):")
    print("Available items and prices:")
    for item, price in item_prices.items():
        print(f"{item}: ${price:.2f}")
    print("-" * 20)

    total_cost = 0

    while True:
        item_name = input("Enter item name: ").strip().lower()
        if item_name == "done":
            break
        if item_name not in item_prices:
            print(f"Sorry, '{item_name}' is not on the menu.")
            continue

        try:
            quantity = int(input(f"Enter quantity for {item_name}: ").strip())
            if quantity <= 0:
                print("Quantity must be a positive number.")
                continue

            if item_name in items:
                items[item_name] += quantity
            else:
                items[item_name] = quantity
            total_cost += item_prices[item_name] * quantity

        except ValueError:
            print("Invalid quantity. Please enter a number.")

    print("\n--- Your Bill ---")
    if not items:
        print("No items added to the bill.")
```

```
total_cost = 0

while True:
    item_name = input("Enter item name: ").strip().lower()
    if item_name == "done":
        break
    if item_name not in item_prices:
        print(f"Sorry, '{item_name}' is not on the menu.")
        continue

    try:
        quantity = int(input(f"Enter quantity for {item_name}: ").strip())
        if quantity <= 0:
            print("Quantity must be a positive number.")
            continue

        if item_name in items:
            items[item_name] += quantity
        else:
            items[item_name] = quantity
        total_cost += item_prices[item_name] * quantity

    except ValueError:
        print("Invalid quantity. Please enter a number.")

print("\n--- Your Bill ---")
if not items:
    print("No items added to the bill.")
```

```
Here's the explanation:

• def generate_bill(): : This line defines a function named generate_bill . All the code inside this function will be executed when the function is called.
• """Prompts the user for item names and quantities, calculates total, and generates a simple bill.""" : This is a docstring, which explains what the function does.
• items = {} : This line initializes an empty dictionary called items . This dictionary will store the items the user orders and their corresponding quantities.
• item_prices = { ... } : This line initializes a dictionary called item_prices . It stores the price for each item. The keys are the item names (strings), and the values are their prices (integers).
• print("Enter item names and quantities (type 'done' when finished):") : This line prints a message to the user, instructing them on how to input items and how to finish the process.
• print("Available items and prices:") : This line prints a heading for the list of available items and their prices.
• for item, price in item_prices.items(): : This line starts a for loop that iterates through each key-value pair in the item_prices dictionary. In each iteration, item will hold the item name, and price will hold its price.
• print(f"{item}: ${price:.2f}") : Inside the loop, this line prints each item name and its price, formatted to two decimal places.
• print("-" * 20) : This line prints a separator line of 20 hyphens.
• total_cost = 0 : This line initializes a variable total_cost to 0. This variable will keep track of the total cost of the ordered items.
• while True: : This line starts an infinite while loop. The loop will continue to execute until a break statement is encountered.
• item_name = input("Enter item name: ").strip().lower() : This line prompts the user to enter an item name using the input() function. .strip() removes any leading or trailing whitespace from the input, and .lower() converts the input to lowercase for case-insensitive matching.
• if item_name == "done": : This line checks if the user entered 'done'.
• break : If the user entered done, this line breaks out of the while loop, ending the input process.
• if item_name not in item_prices: : This line checks if the entered item_name is not a key in the item_prices dictionary.
• print(f"Sorry, '{item_name}' is not on the menu.") : If the item is not found in item_prices, this line prints a message indicating that the item is not on the menu.
• continue : If the item is not on the menu, this line skips the rest of the current loop iteration and goes back to the beginning of the while loop to prompt for the next item.
• try: : This line starts a try block, which is used for error handling. The code inside the try block will be attempted, and if an error occurs, the code in the corresponding except block will be executed.
• quantity = int(input(f"Enter quantity for {item_name}: ").strip()) : Inside the try block, this line prompts the user to enter the quantity for the current item_name. input() gets the user's input, .strip() removes whitespace, and int() attempts to convert the input to an integer.
• if quantity <= 0: : This line checks if the entered quantity is less than or equal to 0.
• print("Quantity must be a positive number.") : If the quantity is not positive, this line prints an error message.
• continue : If the quantity is not positive, this line skips the rest of the current loop iteration.
• if item_name in items: : This line checks if the item_name is already a key in the items dictionary (meaning the user has already ordered this item).
• items[item_name] += quantity : If the item is already in items, this line adds the new quantity to the existing quantity for that item.
• else: : If the item is not yet in items.
```

```
• continue : If the quantity is not positive, this line skips the rest of the current loop iteration.
• if item_name in items: : This line checks if the item_name is already a key in the items dictionary (meaning the user has already ordered this item).
• items[item_name] += quantity : If the item is already in items, this line adds the new quantity to the existing quantity for that item.
• else: : If the item is not yet in items.
• items[item_name] = quantity : This line adds the item_name as a new key to the items dictionary with the entered quantity as its value.
• total_cost += item_prices[item_name] * quantity : This line calculates the cost for the current item (price multiplied by quantity) and adds it to the total_cost.
• except ValueError: : This line starts an except block that will be executed if a ValueError occurs within the try block. This typically happens if the user enters something that cannot be converted to an integer for the quantity.
• print("Invalid quantity. Please enter a number.") : If a ValueError occurs, this line prints an error message.
• print("\n--- Your Bill ---") : After the while loop finishes (when the user enters 'done'), this line prints a heading for the bill.
• if not items: : This line checks if the items dictionary is empty (meaning no items were ordered).
• print("No items added to the bill.") : If items is empty, this line prints a message indicating that no items were added.
• else: : If items is not empty.
• for item, quantity in items.items(): : This line starts a for loop to iterate through the ordered items and their quantities in the items dictionary.
• price = item_prices[item] : Inside the loop, this line retrieves the price of the current item from the item_prices dictionary.
• print(f"{item.capitalize():15} {quantity} x ${price:.2f} = ${quantity * price:.2f}") : This line prints the details for each ordered item, including the item name (capitalized), quantity, price per item, and the total cost for that item, formatted to two decimal places.
• print("-" * 20) : This line prints a separator line.
• print(f"Total: ${total_cost:.2f}") : This line prints the final calculated total_cost, formatted to two decimal places.
• print("-----") : This line prints a closing separator line.
• generate_bill() : This line calls the generate_bill() function to start the program execution.

Let me know if you would like an explanation of any other code!
```

Task 5:

The screenshot shows a Jupyter Notebook environment with a dark theme. The main area displays a Python function `convert_temperature` that takes a temperature value, its unit, and the target unit as input and returns the converted value. The function includes comments for its purpose, arguments, and return values, as well as error handling for invalid units. To the right, a Gemini AI chat window is open, showing a conversation where the user asks for a Python function to convert temperatures, and the AI provides the code and suggestions for further development.

```
def convert_temperature(value: float, from_unit: str, to_unit: str) -> float:
    """
    Converts a temperature value between Celsius ('C'), Fahrenheit ('F'), and Kelvin ('K').

    Args:
        value: The temperature value to convert.
        from_unit: The unit of the input temperature ('C', 'F', or 'K').
        to_unit: The desired unit for the output temperature ('C', 'F', or 'K').

    Returns:
        The converted temperature value.

    Raises:
        ValueError: If the input units are invalid.
    """
    # Convert to Celsius first as an intermediate step
    if from_unit == 'C':
        celsius = value
    elif from_unit == 'F':
        celsius = (value - 32) * 5/9
    elif from_unit == 'K':
        celsius = value - 273.15
    else:
        raise ValueError(f"Invalid 'from_unit': {from_unit}. Must be 'C', 'F', or 'K'.")

    # Convert from Celsius to the desired unit
    if to_unit == 'C':
        return celsius
    elif to_unit == 'F':
        fahrenheit = (celsius * 9/5) + 32
        return fahrenheit
    elif to_unit == 'K':
        kelvin = celsius + 273.15
        return kelvin
    else:
        raise ValueError(f"Invalid 'to_unit': {to_unit}. Must be 'C', 'F', or 'K'.")

# Example Usage:
celsius_temp = 25.0
fahrenheit_temp = convert_temperature(celsius_temp, 'C', 'F')
print(f"{celsius_temp}°C is {fahrenheit_temp:.2f}°F")

fahrenheit_temp = 77.0
celsius_temp_from_f = convert_temperature(fahrenheit_temp, 'F', 'C')
print(f"{fahrenheit_temp}°F is {celsius_temp_from_f:.2f}°C")

celsius_temp = 0.0
kelvin_temp = convert_temperature(celsius_temp, 'C', 'K')
print(f"{celsius_temp}°C is {kelvin_temp:.2f}K")

kelvin_temp = 300.15
celsius_temp_from_k = convert_temperature(kelvin_temp, 'K', 'C')
print(f"{kelvin_temp}K is {celsius_temp_from_k:.2f}°C")

fahrenheit_temp = 212.0
kelvin_temp_from_f = convert_temperature(fahrenheit_temp, 'F', 'K')
print(f"{fahrenheit_temp}°F is {kelvin_temp_from_f:.2f}K")

kelvin_temp = 373.15
fahrenheit_temp_from_k = convert_temperature(kelvin_temp, 'K', 'F')
print(f"{kelvin_temp}K is {fahrenheit_temp_from_k:.2f}°F")

# Example of invalid unit
try:
    convert_temperature(100, 'X', 'C')
except ValueError as e:
    print(f"Error: {e}")

try:
    convert_temperature(100, 'C', 'Y')
except ValueError as e:
    print(f"Error: {e}")
```

Gemini

Write a Python function `convert_temperature(value: float, from_unit: str, to_unit: str) -> float` that converts a temperature value between Celsius (°C), Fahrenheit (°F), and Kelvin (K).

Okay, I can provide a Python function to convert temperatures between Celsius, Fahrenheit, and Kelvin. Here is the code:

I have provided the `convert_temperature` function and examples of how to use it.

Here are a few things you could do next:

- **Build a simple command-line interface (CLI):** You could write a script that prompts the user for the temperature value, the `from_unit`, and the `to_unit`, and then uses the function to perform the conversion and print the result.
- **Create a GUI application:** You could use a library like Tkinter or PyQt to build a graphical interface where users can enter the temperature and select the units from dropdown menus.
- **Integrate with a larger application:** You could incorporate this function into a larger program that deals with weather data, scientific calculations, or any application where temperature conversion is needed.
- **Add more units:** Extend the function to handle other temperature units like Rankine or Réaumur if needed.
- **Add input validation within the CLI/GUI:** Implement more robust validation to ensure the user enters valid numerical values and selects valid units.

What can I help you build?

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

The screenshot shows the same Jupyter Notebook environment, but now the code from the previous block has been executed. The output of the function calls is displayed below the code cells. The output shows the conversion of 25.0°C to 77.00°F, 77.0°F to 25.00°C, 0.0°C to 273.15K, 300.15K to 27.00°C, 212.0°F to 373.15K, and 373.15K to 212.00°F. It also shows the error messages for invalid units 'X' and 'Y'.

```
# Example Usage:
celsius_temp = 25.0
fahrenheit_temp = convert_temperature(celsius_temp, 'C', 'F')
print(f"{celsius_temp}°C is {fahrenheit_temp:.2f}°F")

fahrenheit_temp = 77.0
celsius_temp_from_f = convert_temperature(fahrenheit_temp, 'F', 'C')
print(f"{fahrenheit_temp}°F is {celsius_temp_from_f:.2f}°C")

celsius_temp = 0.0
kelvin_temp = convert_temperature(celsius_temp, 'C', 'K')
print(f"{celsius_temp}°C is {kelvin_temp:.2f}K")

kelvin_temp = 300.15
celsius_temp_from_k = convert_temperature(kelvin_temp, 'K', 'C')
print(f"{kelvin_temp}K is {celsius_temp_from_k:.2f}°C")

fahrenheit_temp = 212.0
kelvin_temp_from_f = convert_temperature(fahrenheit_temp, 'F', 'K')
print(f"{fahrenheit_temp}°F is {kelvin_temp_from_f:.2f}K")

kelvin_temp = 373.15
fahrenheit_temp_from_k = convert_temperature(kelvin_temp, 'K', 'F')
print(f"{kelvin_temp}K is {fahrenheit_temp_from_k:.2f}°F")

# Example of invalid unit
try:
    convert_temperature(100, 'X', 'C')
except ValueError as e:
    print(f"Error: {e}")

try:
    convert_temperature(100, 'C', 'Y')
except ValueError as e:
    print(f"Error: {e}")
```

25.0°C is 77.00°F
77.0°F is 25.00°C
0.0°C is 273.15K
300.15K is 27.00°C
212.0°F is 373.15K
373.15K is 212.00°F
Error: Invalid 'from_unit': X. Must be 'C', 'F', or 'K'.
Error: Invalid 'to_unit': Y. Must be 'C', 'F', or 'K'.

Explanation:

Gemini

Here's the line-by-line explanation:

- `def convert_temperature(value: float, from_unit: str, to_unit: str) -> float:` This line defines a function named `convert_temperature`. It takes three arguments: `value` (a float representing the temperature value), `from_unit` (a string representing the original unit), and `to_unit` (a string representing the target unit). The `-> float` indicates that the function is expected to return a float.
- `""" ... """` This is a docstring that explains what the function does, its arguments (`Args`), what it returns (`Returns`), and any potential errors it might raise (`Raises`).
- `# Convert to Celsius first as an intermediate step` This is a comment explaining the approach taken in the following lines. Converting to Celsius first simplifies the conversion process as you only need conversion formulas to and from Celsius.
- `if from_unit == 'C':` This line checks if the `from_unit` is 'C' (Celsius).
- `celsius = value` If the `from_unit` is 'C', the `celsius` variable is assigned the original `value` directly.
- `elif from_unit == 'F':` If the `from_unit` is not 'C', this line checks if it's 'F' (Fahrenheit).
- `celsius = (value - 32) * 5/9` If the `from_unit` is 'F', this line converts the `Fahrenheit` `value` to Celsius using the standard formula.
- `elif from_unit == 'K':` If the `from_unit` is not 'C' or 'F', this line checks if it's 'K' (Kelvin).
- `celsius = value - 273.15` If the `from_unit` is 'K', this line converts the `Kelvin` `value` to Celsius.
- `else:` If the `from_unit` is none of the recognized units ('C', 'F', or 'K').
- `raise ValueError(f"Invalid 'from_unit': {from_unit}. Must be 'C', 'F', or 'K'.")` This line raises a `ValueError` with an informative message indicating that the provided `from_unit` is invalid.
- `# Convert from Celsius to the desired unit` This is a comment explaining the next set of lines.
- `if to_unit == 'C':` This line checks if the `to_unit` is 'C'.
- `return celsius` If the `to_unit` is 'C', the function returns the `celsius` value (which was already calculated in the previous step).
- `elif to_unit == 'F':` If the `to_unit` is not 'C', this line checks if it's 'F'.
- `fahrenheit = (celsius * 9/5) + 32` If the `to_unit` is 'F', this line converts the `celsius` value to Fahrenheit using the standard formula.
- `return fahrenheit` The function returns the calculated `fahrenheit` value.
- `elif to_unit == 'K':` If the `to_unit` is not 'C' or 'F', this line checks if it's 'K'.
- `kelvin = celsius + 273.15` If the `to_unit` is 'K', this line converts the `celsius` value to Kelvin.
- `return kelvin` The function returns the calculated `kelvin` value.
- `else:` If the `to_unit` is none of the recognized units ('C', 'F', or 'K').
- `raise ValueError(f"Invalid 'to_unit': {to_unit}. Must be 'C', 'F', or 'K'.")` This line raises a `ValueError` indicating that the provided `to_unit` is invalid.
- `# Example Usage:` This is a comment marking the start of the example usage section.
- The following lines demonstrate how to call the `convert_temperature` function with different values and units and print the results, showcasing various conversion scenarios.
- `try:` and `except ValueError as e:` These blocks demonstrate how to handle the `ValueError` that the function might raise if invalid units are provided. The `try` block attempts the conversion, and if a `ValueError` occurs, the `except` block catches it and prints an error message.

This explanation covers the functionality of each line in the `convert_temperature` function and the example usage. Let me know if you have any more questions!

What can I help you build?

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

5:56 PM

Python 3