2403A51272

Anjali Maripudi

QUESTION1:

Create a Python function that converts an amount from one currency to another using exchange rates stored in a dictionary. Use GitHub Copilot along with VS Code. Use Few shot prompting.

FEW-SHOT PROMPT GIVEN TO COLAB:

Create a Python function that converts an amount from one currency to another using exchange rates stored in a dictionary.

FOR EXAMPLE:

# The exchange rates are stored in a dictionary with currency codes as keys.

# Example 1: # convert\_currency(100, "USD", "INR", exchange\_rates) -> 8310.0

# Example 2: # convert\_currency(50, "EUR", "USD", exchange\_rates) -> ~54.35

# Example 3: # convert\_currency(1000, "JPY", "GBP", exchange\_rates) -> ~5.36

CODE GIVEN BY COLAB:

def convert\_currency(amount: float, from\_currency: str, to\_currency: str, rates: dict) -> float:

    """Converts an amount from one currency to another using exchange rates.

    Args:

        amount: The amount to convert.

        from\_currency: The currency to convert from (e.g., "USD").

        to\_currency: The currency to convert to (e.g., "INR").

        rates: A dictionary of exchange rates, where keys are currency pairs

               (e.g., "USD\_INR") and values are the exchange rates.

    Returns:

        The converted amount.

    """

    if from\_currency == to\_currency:

        return amount

    # Construct the currency pair key

    currency\_pair = f"{from\_currency}\_{to\_currency}"

    if currency\_pair in rates:

        return amount \* rates[currency\_pair]

    else:

        # Handle cases where the direct exchange rate is not available

        # For simplicity, this example assumes direct rates are provided.

        # A more robust solution would involve finding intermediate conversions.

        raise ValueError(f"Exchange rate for {currency\_pair} not found.")

# Example usage:

exchange\_rates = {

    "USD\_INR": 83.10,

    "EUR\_USD": 1.087,

    "JPY\_GBP": 0.00536,

    "USD\_EUR": 1/1.087, # Adding the inverse rate

    "INR\_USD": 1/83.10, # Adding the inverse rate

    "GBP\_JPY": 1/0.00536 # Adding the inverse rate

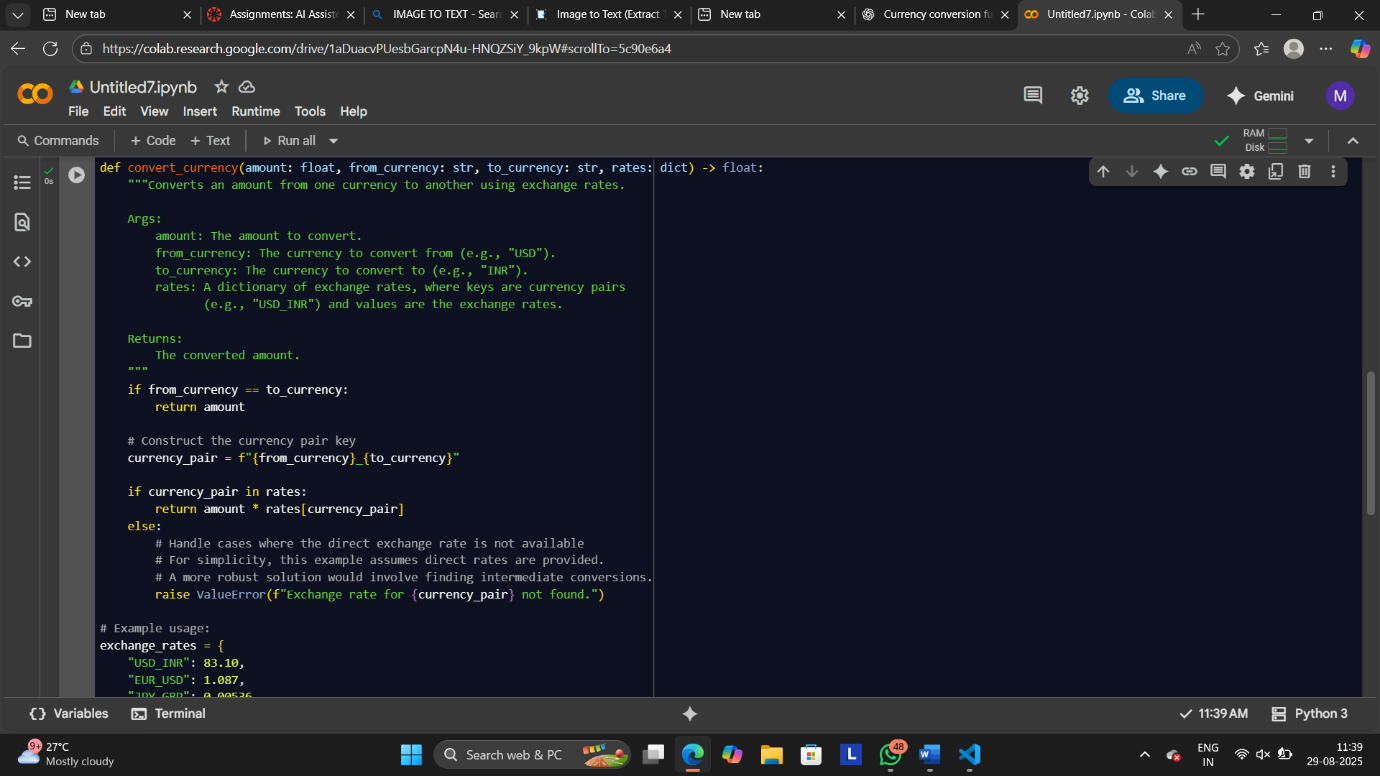
}

print(f"100 USD to INR: {convert\_currency(100, 'USD', 'INR', exchange\_rates)}")

print(f"50 EUR to USD: {convert\_currency(50, 'EUR', 'USD', exchange\_rates)}")

print(f"1000 JPY to GBP: {convert\_currency(1000, 'JPY', 'GBP', exchange\_rates)}")

print(f"100 INR to USD: {convert\_currency(100, 'INR', 'USD', exchange\_rates)}")



A screenshot of a computer

AI-generated content may be incorrect.

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

QUESTION2:

Write a Python program to extract all email addresses from a block of text using regular expressions. GitHub Copilot along with VS Code. Use zero shot prompting.

ZERO-SHOT PROMPT GIVEN TO COLAB:

Write a Python program to extract all email addresses from a block of text using regular expressions.

CODE GIVEN BY COLAB:

def convert\_currency(amount: float, from\_currency: str, to\_currency: str, rates: dict) -> float:

    """Converts an amount from one currency to another using exchange rates.

    Args:

        amount: The amount to convert.

        from\_currency: The currency to convert from (e.g., "USD").

        to\_currency: The currency to convert to (e.g., "INR").

        rates: A dictionary of exchange rates, where keys are currency pairs

               (e.g., "USD\_INR") and values are the exchange rates.

    Returns:

        The converted amount.

    """

    if from\_currency == to\_currency:

        return amount

    # Construct the currency pair key

    currency\_pair = f"{from\_currency}\_{to\_currency}"

    if currency\_pair in rates:

        return amount \* rates[currency\_pair]

    else:

        # Handle cases where the direct exchange rate is not available

        # For simplicity, this example assumes direct rates are provided.

        # A more robust solution would involve finding intermediate conversions.

        raise ValueError(f"Exchange rate for {currency\_pair} not found.")

# Example usage:

exchange\_rates = {

    "USD\_INR": 83.10,

    "EUR\_USD": 1.087,

    "JPY\_GBP": 0.00536,

    "USD\_EUR": 1/1.087, # Adding the inverse rate

    "INR\_USD": 1/83.10, # Adding the inverse rate

    "GBP\_JPY": 1/0.00536 # Adding the inverse rate

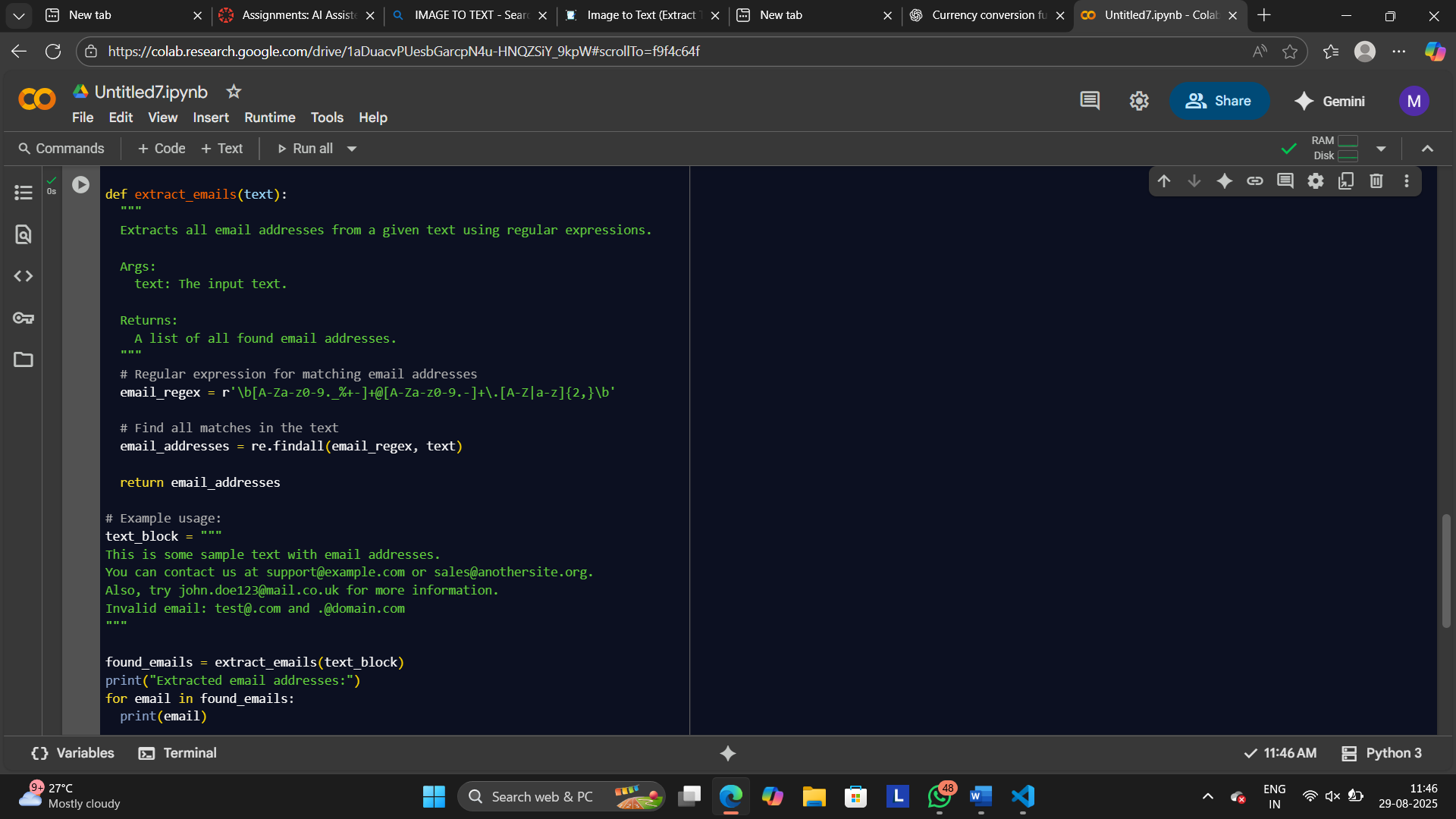
}

print(f"100 USD to INR: {convert\_currency(100, 'USD', 'INR', exchange\_rates)}")

print(f"50 EUR to USD: {convert\_currency(50, 'EUR', 'USD', exchange\_rates)}")

print(f"1000 JPY to GBP: {convert\_currency(1000, 'JPY', 'GBP', exchange\_rates)}")

print(f"100 INR to USD: {convert\_currency(100, 'INR', 'USD', exchange\_rates)}")



OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

QUESTION3:

Given a list of movies with their genres, Write a Python function that recommends movies based on a user's preferred genre. Use the Cursor AI tool. Use few shot prompting.

FEW-SHOT PROMPT GIVEN TO CURSOR:

Write a Python function that recommends movies based on a user's preferred genre.

FOR EXAMPLE:

# Example 1: # movies = [("Inception", "Sci-Fi"), ("Titanic", "Romance"), ("The Matrix", "Sci-Fi")] # recommend\_movies("Sci-Fi", movies) -> ["Inception", "The Matrix"] #

# Example 2: # movies = [("Avengers", "Action"), ("The Notebook", "Romance"), ("Die Hard", "Action")] # recommend\_movies("Action", movies) -> ["Avengers", "Die Hard"] #

# Example 3: # movies = [("Coco", "Animation"), ("Frozen", "Animation"), ("Interstellar", "Sci-Fi")] # recommend\_movies("Animation", movies) -> ["Coco", "Frozen"] def recommend\_movies(genre: str, movies: list) -> list: pass

CODE GIVEN BY CURSOR:

def recommend\_movies(*genre*: str, *movies*: list) -> list:

    """

    Recommends movies based on a user's preferred genre.

    Args:

        genre (str): The preferred genre to filter movies by

        movies (list): List of (movie\_name, genre) tuples

    Returns:

        list: List of movie names that match the preferred genre

    """

    recommended = []

*for* movie\_name, movie\_genre *in* *movies*:

*if* movie\_genre == *genre*:

            recommended.append(movie\_name)

*return* recommended

*# Test cases*

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

*# Example 1*

    movies1 = [("Inception", "Sci-Fi"), ("Titanic", "Romance"), ("The Matrix", "Sci-Fi")]

    print("Example 1:")

    print(f"movies = {movies1}")

    print(f'recommend\_movies("Sci-Fi", movies) -> {recommend\_movies("Sci-Fi", movies1)}')

    print()

*# Example 2*

    movies2 = [("Avengers", "Action"), ("The Notebook", "Romance"), ("Die Hard", "Action")]

    print("Example 2:")

    print(f"movies = {movies2}")

    print(f'recommend\_movies("Action", movies) -> {recommend\_movies("Action", movies2)}')

    print()

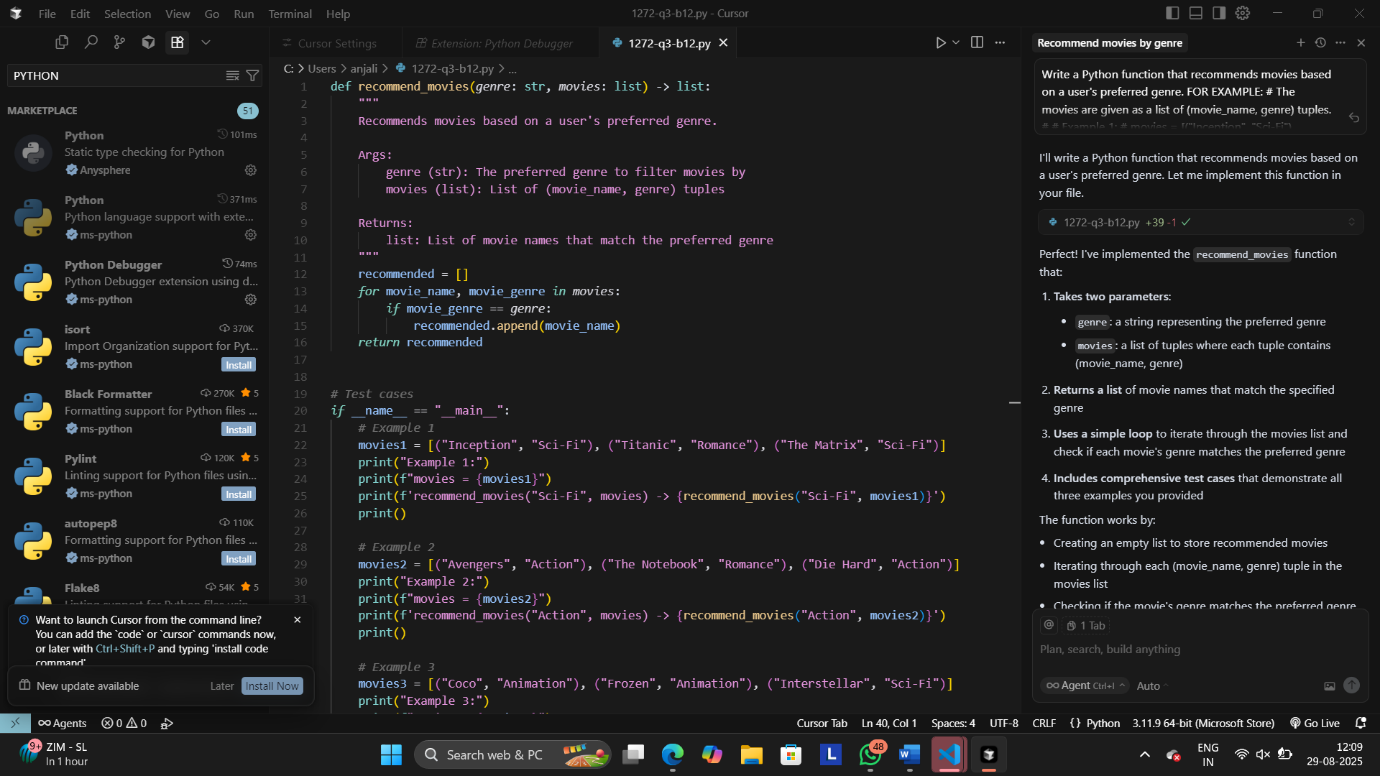
*# Example 3*

    movies3 = [("Coco", "Animation"), ("Frozen", "Animation"), ("Interstellar", "Sci-Fi")]

    print("Example 3:")

    print(f"movies = {movies3}")

    print(f'recommend\_movies("Animation", movies) -> {recommend\_movies("Animation", movies3)}')



OUTPUT:

A screenshot of a computer program

AI-generated content may be incorrect.

QUESTION4:

Write Python code that reads a CSV file containing student names and marks in 3 subjects. Calculate the total and average marks for each student. Use the Cursor AI tool.

PROMPT GIVEN TO CURSOR:

Write Python code that reads a CSV file containing student names and marks in 3 subjects. Calculate the total and average marks for each student.

CODE GIVEN BY CURSOR:

*import* csv

*import* sys

*from* pathlib *import* Path

*from* typing *import* List, Tuple

def parse\_row\_to\_marks(*row*: List[str]) -> Tuple[str, List[float]]:

    """

    Accept a raw list for a CSV row where the first column is the student name

    and the next three columns are numeric marks. Returns (name, [m1, m2, m3]).

    """

*if* len(*row*) < 4:

*raise* ValueError("Each row must have at least 4 columns: name and 3 marks")

    name = *row*[0].strip()

*try*:

        marks = [float(*row*[1]), float(*row*[2]), float(*row*[3])]

*except* (ValueError, IndexError):

*raise* ValueError(f"Invalid marks for student '{name}'. Ensure three numeric marks.")

*return* name, marks

def read\_students\_from\_csv(*csv\_path*: Path) -> List[Tuple[str, List[float]]]:

    """

    Reads students and their three marks from a CSV. Supports both header and no-header formats.

    - Header example: Name,Sub1,Sub2,Sub3

    - No header example: Alice,78,82,91

    Returns a list of (name, [m1, m2, m3]).

    """

    students: List[Tuple[str, List[float]]] = []

*with* *csv\_path*.open(*newline*="", *encoding*="utf-8-sig") *as* f:

        sample = f.read(1024)

        f.seek(0)

*try*:

            dialect = csv.Sniffer().sniff(sample)

*except* csv.Error:

            dialect = csv.excel

*try*:

            has\_header = csv.Sniffer().has\_header(sample)

*except* csv.Error:

            has\_header = True

        reader = csv.reader(f, dialect)

        first = True

*for* row *in* reader:

*if* not row or all(c.strip() == "" *for* c *in* row):

*continue*

*if* first and has\_header:

                first = False

*# Try to detect header by names; if header is actually data, fall through*

                lowered = [c.strip().lower() *for* c *in* row]

*if* any(h *in* lowered *for* h *in* ["name", "student", "student name", "sname"]):

*# Looks like a header; skip it*

*continue*

*# If not obviously a header, process it as data*

            first = False

            name, marks = parse\_row\_to\_marks(row)

            students.append((name, marks))

*return* students

def compute\_totals\_and\_averages(*students*: List[Tuple[str, List[float]]]):

    results = []

*for* name, marks *in* *students*:

        total = sum(marks)

        average = total / 3.0

        results.append((name, marks[0], marks[1], marks[2], total, average))

*return* results

def write\_results\_to\_csv(*results*, *output\_path*: Path) -> None:

*with* *output\_path*.open("w", *newline*="", *encoding*="utf-8") *as* f:

        writer = csv.writer(f)

        writer.writerow(["Name", "Sub1", "Sub2", "Sub3", "Total", "Average"])

*for* rec *in* *results*:

            name, m1, m2, m3, total, avg = rec

            writer.writerow([name, m1, m2, m3, total, round(avg, 2)])

def main() -> None:

*# Get input CSV path from argv or prompt*

*if* len(sys.argv) >= 2:

        input\_path = Path(sys.argv[1]).expanduser()

*else*:

*try*:

            input\_str = input("Enter path to input CSV: ").strip()

*except* EOFError:

            input\_str = ""

        input\_path = Path(input\_str).expanduser() *if* input\_str *else* Path("students.csv")

*if* not input\_path.exists():

        print(f"Input file not found: {input\_path}")

        print("Expected CSV columns: name, mark1, mark2, mark3")

*return*

*try*:

        students = read\_students\_from\_csv(input\_path)

*except* Exception *as* exc:

        print(f"Failed to read CSV: {exc}")

*return*

*if* not students:

        print("No student records found.")

*return*

    results = compute\_totals\_and\_averages(students)

*# Print to console*

    print("Name, Sub1, Sub2, Sub3, Total, Average")

*for* name, m1, m2, m3, total, avg *in* results:

        print(f"{name}, {m1}, {m2}, {m3}, {total}, {avg:.2f}")

*# Write output CSV next to input*

    output\_path = input\_path.with\_name(input\_path.stem + "\_with\_totals.csv")

*try*:

        write\_results\_to\_csv(results, output\_path)

        print(f"Output written to: {output\_path}")

*except* Exception *as* exc:

        print(f"Failed to write output CSV: {exc}")

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

    main()

A screenshot of a computer program

AI-generated content may be incorrect.

OUTPUT:

A black screen with white text

AI-generated content may be incorrect.