

AI Assisted Coding

Name : PRAVALIKA MUTHOJU

Hall : 2303a52347

Batch : 34

Lab 3: Prompt Engineering – Improving Prompts and Context Management

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.

```
assignment3.2.py
10 def calculator(a=None, b=None, op=None):
    print("Invalid operator. Choose one of: + - * /")
    if a is None:
        a = _get_number("Enter the first number: ")
    if b is None:
        b = _get_number("Enter the second number: ")
    if op is None:
        op = _get_operator("Enter an operator (+ - * /): ")

    try:
        a = float(a)
        b = float(b)
    except Exception:
        print("Error: Invalid numeric input.")
        return None

    if op == '+':
        result = a + b
    elif op == '-':
        result = a - b
    elif op == '*':
        result = a * b
    elif op == '/':
        result = a / b
    else:
        print("Unknown operator: ", op)
        return None

    return result
```

The screenshot shows a code editor interface with the AI Assistant extension. On the left, there's a file named 'assignment3.2.py' containing Python code for a calculator function. The code handles addition, subtraction, multiplication, and division. On the right, there's a sidebar titled 'AI Assistant' which tracks the AI's progress and provides context for the code. It includes a 'todo' list and a 'history' section showing previous interactions.

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.

The screenshot shows the AI Assistant IDE interface. In the top right, there's an AI interface window titled "PYTHON CALCULATOR FUNCTION WITH ERROR HANDLING" with a task description: "Write a Python function that takes a list of student marks (integers between 0 and 100) and returns the marks sorted in descending order using an efficient sorting method." Below it, the code editor shows a Python script named "marks_utils.py". The code defines a function `sort_marks_desc` that uses Python's built-in `sorted` function (Timsort) for efficiency. It includes validation for input type and range. A quick manual test at the bottom prints the sorted marks for a list [56, 90, 100, 8, 75]. The terminal below shows the command `python -u "c:/Users/95425/Downloads/AiAssistant/marks_utils.py"` and its output: [[100, 90, 75, 56, 8]]. The bottom right shows a "Code" tab and a terminal status bar.

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.

The screenshot shows the AI Assistant IDE interface. In the top right, there's an AI interface window titled "PYTHON CALCULATOR FUNCTION WITH ERROR HANDLING" with a task description: "Create an efficient Python function to check whether a given number is prime. Handle edge cases such as 0, 1, and negative numbers." Below it, the code editor shows a Python script named "marks_utils.py". The code defines a function `is_prime` that efficiently checks divisibility up to sqrt(n). It handles edge cases like 0, 1, and negative numbers. A quick manual test at the bottom prints the prime status for a list of values: 2, 4, 1, -3, 17, 18. The terminal below shows the command `python -u "c:/Users/95425/Downloads/AiAssistant/marks_utils.py"` and its output: Input: 2 -> Output: Prime, Input: 4 -> Output: Not Prime, Input: 1 -> Output: Not Prime, Input: -3 -> Output: Not Prime, Input: 17 -> Output: Prime, Input: 18 -> Output: Not Prime. The bottom right shows a "Code" tab and a terminal status bar.

Task Description-4

- Prompt-Guided UI Design for Student Grading System: Create a user interface for a student grading system that calculates total marks, percentage, and grade based on user input.

File Edit Selection View Go Run Terminal Help

grading_system.py

```
def grade_from_percentage(pct):
    """Return grade letter for percentage using given mapping.
    A: >=90, B: >=75, C: >=60, D: <60

    try:
        pct = float(pct)
    except Exception:
        return 'Invalid'
    if pct >= 90:
        return 'A'
    if pct >= 75:
        return 'B'
    if pct >= 60:
        return 'C'
    return 'D'

def grade_student(name, marks):
    """Compute total, percentage, and grade for a student.

    'marks' may be a dict mapping subject->mark or an iterable of numeric marks.
    Returns a dict with 'name', 'marks' (as dict), 'total', 'percentage', 'grade'.
    Percentage is calculated as the average mark (treating each subject as out of 100).
    """
    # Normalize marks into an ordered dict-like mapping
    if isinstance(marks, dict):
        subj_marks = dict(marks)
    else:
        # assume iterable of marks without subject names
        subj_marks = {f'Subj {i+1}': float(m) for i, m in enumerate(marks)}
    # Validate numeric marks and compute total
```

AIAssistant

PYTHON CALCULATOR FUNCTION WITH ERROR HANDLING

I'll add a new `grading_system.py` with interactive CLI functions plus a demo, then run a quick non-interactive demo to verify output.

- Edited `grading_system.py -108-1`
- Created 6 nodes
- `python -c "import grading_system; grading_system.demo()"`
- Checked background terminal output

Progress added: `grading_system.py` with interactive CLI `calculate_results`, `assign_grade`, and a non-interactive `demo()`.

- File: `grading_system.py1-200`
- How to run demo
- To run interactively:

`python grading_system.py`

Would you like a pytest test file for these functions or integration into an existing script?

Decided on coding response and formatted code

I'll open the two files and paste their contents here.

Investigated file issues and read files

- `grading_system.py`: `grading_system.py1-200` — contains the grading CLI functions (`calculate_results`, `assign_grade`, `display_results`, `create_demo`, `run_interactive`, `demo`).
- `marks_utils.py`: `marks_utils1` — currently empty.

Shall I restore the `__init__.py` and `setup_wkns_desc` implementations into `marks_utils` and paste the code here?

GPT-5 min + tx

grading_system.py

Describe what to build next

Agent Auto

File Edit Selection View Go Run Terminal Help

grading_system.py

```
def grade_student(name, marks):
    # Validate numeric marks and compute total
    total = 0.0
    count = 0
    for k, v in list(subj_marks.items()):
        try:
            mv = float(v)
        except Exception:
            mv = 0.0
        subj_marks[k] = mv
        total += mv
        count += 1

    if count == 0:
        percentage = 0.0
    else:
        average = total / count
        percentage = average # treating average as percentage (out of 100)

    grade = grade_from_percentage(percentage)
    return {
        'name': name,
        'marks': subj_marks,
        'total': total,
        'percentage': round(percentage, 2),
        'grade': grade,
    }

def interactive_grading_system():
    """Interactively accept multiple students and their subject marks.
    For each student:
```

AIAssistant

PYTHON CALCULATOR FUNCTION WITH ERROR HANDLING

I'll add a new `grading_system.py` with interactive CLI functions plus a demo, then run a quick non-interactive demo to verify output.

- Edited `grading_system.py -108-4`
- Created 6 nodes
- `python -c "import grading_system; grading_system.demo()"`
- Checked background terminal output

Progress added: `grading_system.py` with interactive CLI `calculate_results`, `assign_grade`, and a non-interactive `demo()`.

- File: `grading_system.py1-200`
- How to run demo
- To run interactively:

`python grading_system.py`

Would you like a pytest test file for these functions or integration into an existing script?

Decided on coding response and formatted code

I'll open the two files and paste their contents here.

Investigated file issues and read files

- `grading_system.py`: `grading_system.py1-200` — contains the grading CLI functions (`calculate_results`, `assign_grade`, `display_results`, `create_demo`, `run_interactive`, `demo`).
- `marks_utils.py`: `marks_utils1` — currently empty.

Shall I restore the `__init__.py` and `setup_wkns_desc` implementations into `marks_utils` and paste the code here?

GPT-5 min + tx

grading_system.py

Describe what to build next

Agent Auto

The screenshot shows the AI Assistant interface with the grading_system.py file open. The code defines an interactive grading system that prompts for student names and subjects, validates marks, and calculates totals, percentages, and grades. A detailed analysis pane on the right provides insights into the code's structure and potential improvements.

```
def interactive_grading_system():
    """Interactively accept multiple students and their subject marks.

    For each student:
    - Prompt for student name (blank to finish)
    - Prompt repeatedly for subject name and mark (blank subject to finish)
    - Validate numeric marks
    - Compute and display total, percentage, and grade in formatted output

    results = []
    print('Enter student data. Leave student name blank to finish.')
    try:
        while True:
            name = input('\nStudent name: ').strip()
            if name == '':
                break

            subjects = {}
            print("Enter subject name and mark. Leave subject blank to finish.")
            while True:
                subj = input(' Subject name: ').strip()
                if subj == '':
                    break
                mark_raw = input(' Mark (0-100): ').strip()
                try:
                    mark = float(mark_raw)
                except ValueError:
                    print(' Invalid mark - please enter a numeric value.')
                    continue
                subjects[subj] = mark

            rec = grade_student(name, subjects)
            results.append(rec)

    except KeyboardInterrupt:
        print("\n----- Result -----")
        print(f"Student: {rec['name']}")
        print("Subjects:")
        for s, m in rec['marks'].items():
            print(f" {s}: {m}")
        total_possible = len(rec['marks']) * 100 if rec['marks'] else 0
        print(f"Total : {rec['total']} / {total_possible}")
        print(f"Percent : {rec['percentage']}%")
        print(f"Grade : {rec['grade']}")
        print("-----")

    again = input("Add another student? (y/n): ").strip().lower()
    if again not in ('y', 'yes'):
        break
    except KeyboardInterrupt:
        print("\n----- Result -----")
        print(f"Student: {rec['name']}")
        print("Subjects:")
        for s, m in rec['marks'].items():
            print(f" {s}: {m}")
        total_possible = len(rec['marks']) * 100 if rec['marks'] else 0
        print(f"Total : {rec['total']} / {total_possible}")
        print(f"Percent : {rec['percentage']}%")
        print(f"Grade : {rec['grade']}")
        print("-----")

    return results
```

This screenshot shows the same grading_system.py code and analysis as the previous one, but with some additional logic added for displaying results. The code now includes a loop to print student details and calculate totals and percentages for each student.

```
def interactive_grading_system():
    """Interactively accept multiple students and their subject marks.

    For each student:
    - Prompt for student name (blank to finish)
    - Prompt repeatedly for subject name and mark (blank subject to finish)
    - Validate numeric marks
    - Compute and display total, percentage, and grade in formatted output

    results = []
    print('Enter student data. Leave student name blank to finish.')
    try:
        while True:
            name = input('\nStudent name: ').strip()
            if name == '':
                break

            subjects = {}
            print("Enter subject name and mark. Leave subject blank to finish.")
            while True:
                subj = input(' Subject name: ').strip()
                if subj == '':
                    break
                mark_raw = input(' Mark (0-100): ').strip()
                try:
                    mark = float(mark_raw)
                except ValueError:
                    print(' Invalid mark - please enter a numeric value.')
                    continue
                subjects[subj] = mark

            rec = grade_student(name, subjects)
            results.append(rec)

    except KeyboardInterrupt:
        print("\n----- Result -----")
        print(f"Student: {rec['name']}")
        print("Subjects:")
        for s, m in rec['marks'].items():
            print(f" {s}: {m}")
        total_possible = len(rec['marks']) * 100 if rec['marks'] else 0
        print(f"Total : {rec['total']} / {total_possible}")
        print(f"Percent : {rec['percentage']}%")
        print(f"Grade : {rec['grade']}")
        print("-----")

    again = input("Add another student? (y/n): ").strip().lower()
    if again not in ('y', 'yes'):
        break
    except KeyboardInterrupt:
        print("\n----- Result -----")
        print(f"Student: {rec['name']}")
        print("Subjects:")
        for s, m in rec['marks'].items():
            print(f" {s}: {m}")
        total_possible = len(rec['marks']) * 100 if rec['marks'] else 0
        print(f"Total : {rec['total']} / {total_possible}")
        print(f"Percent : {rec['percentage']}%")
        print(f"Grade : {rec['grade']}")
        print("-----")

    return results
```

This screenshot shows the final version of the grading_system.py code, which includes the logic for displaying results and running interactively. The AI Assistant provides a detailed analysis of the code's structure and potential improvements.

```
import math

def interactive_grading_system():
    """Interactively accept multiple students and their subject marks.

    For each student:
    - Prompt for student name (blank to finish)
    - Prompt repeatedly for subject name and mark (blank subject to finish)
    - Validate numeric marks
    - Compute and display total, percentage, and grade in formatted output

    results = []
    print('Enter student data. Leave student name blank to finish.')
    try:
        while True:
            name = input('\nStudent name: ').strip()
            if name == '':
                break

            subjects = {}
            print("Enter subject name and mark. Leave subject blank to finish.")
            while True:
                subj = input(' Subject name: ').strip()
                if subj == '':
                    break
                mark_raw = input(' Mark (0-100): ').strip()
                try:
                    mark = float(mark_raw)
                except ValueError:
                    print(' Invalid mark - please enter a numeric value.')
                    continue
                subjects[subj] = mark

            rec = grade_student(name, subjects)
            results.append(rec)

    except KeyboardInterrupt:
        print("\n----- Result -----")
        print(f"Student: {rec['name']}")
        print("Subjects:")
        for s, m in rec['marks'].items():
            print(f" {s}: {m}")
        total_possible = len(rec['marks']) * 100 if rec['marks'] else 0
        print(f"Total : {rec['total']} / {total_possible}")
        print(f"Percent : {rec['percentage']}%")
        print(f"Grade : {rec['grade']}")
        print("-----")

    again = input("Add another student? (y/n): ").strip().lower()
    if again not in ('y', 'yes'):
        break
    except KeyboardInterrupt:
        print("\n----- Result -----")
        print(f"Student: {rec['name']}")
        print("Subjects:")
        for s, m in rec['marks'].items():
            print(f" {s}: {m}")
        total_possible = len(rec['marks']) * 100 if rec['marks'] else 0
        print(f"Total : {rec['total']} / {total_possible}")
        print(f"Percent : {rec['percentage']}%")
        print(f"Grade : {rec['grade']}")
        print("-----")

    return results
```

Task Description-5

- Analyzing Prompt Specificity in Unit Conversion Functions: Improving a Unit Conversion Function (Kilometers to Miles and Miles to Kilometers) Using Clear Instructions.

```
assignment12.py
grading_system.py
marks_utility.py
conversions.py

# grading_system.py
def convert_distance(value, unit):
    """Convert distance between kilometers and miles.

    Parameters:
        - value: numeric distance (int/float or string parseable to float)
        - unit: string, either 'km' to convert km->miles or 'miles' to convert miles->km

    Returns:
        - converted_value (float rounded to 2 decimals), output_unit (str)

    Raises:
        - ValueError for invalid unit or non-numeric value
    """
    if unit is None:
        raise ValueError('Unit must be specified as "km" or "miles"')
    u = unit.strip().lower()
    try:
        v = float(value)
    except Exception:
        raise ValueError('Distance value must be numeric')

    if u == 'km':
        return (round(v * 0.621371, 2), 'miles')
    if u == 'miles' or u == 'mile':
        return (round(v / 0.621371, 2), 'km')
    raise ValueError(f'Invalid unit: {unit!r}. Use "km" or "miles"')

def interactive_convert_loop():
    """Prompt user for conversions until they choose to exit."""
    print('Unit converter - 1 km = 0.621371 miles')
    try:
        while True:
            raw = input("Enter value and unit (e.g. '10 km' or '5 miles'), or 'q' to quit")
            if raw.lower() in ('q', 'quit', 'exit'):
                print('Goodbye.')
                break
            if not raw:
                continue
            parts = raw.split()
            if len(parts) == 1:
                print("Please provide both value and unit, e.g. '10 km'.")
                continue
            # join all but last as value in case of spaces in numbers (rare)
            value_str = ' '.join(parts[:-1])
            unit = parts[-1]
```

AI Assistant panel content:

- PYTHON CALCULATOR FUNCTION WITH ERROR HANDLING
- grading_system.py : grading_system.py(29) — contains the grading CLI and functions calculate_results, assign_grade, display_results, collect_marks_interactive, demo.
- marks_utility.py : marks_utility.py — currently empty.
- Shall I restore the `__init__.py` and `sort_marks_desc` implementations into `marks_utility.py` and paste the code here?
- Write a Python program that includes two functions `km_to_miles` and `miles_to_km`. Use correct conversion formulas, accept user input, and display accurate results with clear output messages.
- Planned and implemented conversion program functions
- I'll add a small plan then create `conversions.py` with the two functions, CLI input parsing, and a demo.
- Managed and tracked todo items for planning
- Done — added the conversion utility.
- File: `conversions.py(20)`
- Functions: `M km_to_miles(mile: float) -> float`, `M miles_to_km(miles: float) -> float`, with accurate conversion factors and docstrings.
- Run interactively!
- python conversions.py
- Programmatic use
- from conversions import km_to_miles, miles_to_km
print(km_to_miles(5)) # miles for 5 km
print(miles_to_km(5)) # km for 5 miles
- Want me to add unit tests (pytest) for these functions?
- Tests (0/0)
 > 1 file changed +70 -1
 ↗ + w grading_system.py
 Describe what to build next!

```
assignment12.py
grading_system.py
marks_utility.py
conversions.py

# grading_system.py
def convert_distance(value, unit):
    u = unit.strip().lower()
    try:
        v = float(value)
    except Exception:
        raise ValueError('Distance value must be numeric')

    if u == 'km':
        return (round(v * 0.621371, 2), 'miles')
    if u == 'miles' or u == 'mile':
        return (round(v / 0.621371, 2), 'km')
    raise ValueError(f'Invalid unit: {unit!r}. Use "km" or "miles"')

def interactive_convert_loop():
    """Prompt user for conversions until they choose to exit."""
    print('Unit converter - 1 km = 0.621371 miles')
    try:
        while True:
            raw = input("Enter value and unit (e.g. '10 km' or '5 miles'), or 'q' to quit")
            if raw.lower() in ('q', 'quit', 'exit'):
                print('Goodbye.')
                break
            if not raw:
                continue
            parts = raw.split()
            if len(parts) == 1:
                print("Please provide both value and unit, e.g. '10 km'.")
                continue
            # join all but last as value in case of spaces in numbers (rare)
            value_str = ' '.join(parts[:-1])
            unit = parts[-1]
```

AI Assistant panel content:

- PYTHON CALCULATOR FUNCTION WITH ERROR HANDLING
- grading_system.py : grading_system.py(29) — contains the grading CLI and functions calculate_results, assign_grade, display_results, collect_marks_interactive, demo.
- marks_utility.py : marks_utility.py — currently empty.
- Shall I restore the `__init__.py` and `sort_marks_desc` implementations into `marks_utility.py` and paste the code here?
- Write a Python program that includes two functions `km_to_miles` and `miles_to_km`. Use correct conversion formulas, accept user input, and display accurate results with clear output messages.
- Planned and implemented conversion program functions
- I'll add a small plan then create `conversions.py` with the two functions, CLI input parsing, and a demo.
- Managed and tracked todo items for planning
- Done — added the conversion utility.
- File: `conversions.py(20)`
- Functions: `M km_to_miles(mile: float) -> float`, `M miles_to_km(miles: float) -> float`, with accurate conversion factors and docstrings.
- Run interactively!
- python conversions.py
- Programmatic use
- from conversions import km_to_miles, miles_to_km
print(km_to_miles(5)) # miles for 5 km
print(miles_to_km(5)) # km for 5 miles
- Want me to add unit tests (pytest) for these functions?
- Tests (0/0)
 > 1 file changed +70 -1
 ↗ + w grading_system.py
 Describe what to build next!

The screenshot shows a code editor interface with several windows open. The main window displays a Python script named `grading_system.py`. The script contains a function `interactive_convert_loop()` which handles user input for distance conversion. It includes error handling for non-numeric input and attempts to convert the input to a float. The code uses `join` to handle cases where there might be spaces between the value and the unit. It also handles a `KeyboardInterrupt`.

```
def interactive_convert_loop():
    if not raw:
        continue

    parts = raw.split()
    if len(parts) == 1:
        print("Please provide both value and unit, e.g. '10 km'.")
        continue

    # join all but last as value in case of spaces in numbers (rare)
    value_str = ''.join(parts[:-1])
    unit = parts[-1]

    try:
        converted, out_unit = convert_distance(value_str, unit)
    except ValueError as e:
        print('Error:', e)
        continue

    # Format and display
    try:
        val_num = float(value_str)
    except Exception:
        val_num = value_str
    print(f"{val_num} {unit} -> {converted} {out_unit}")

except KeyboardInterrupt:
    print('InInterrupted. Exiting.')

if __name__ == '__main__':
    interactive_convert_loop()
```

On the right side of the interface, there is a sidebar titled "CMD" which provides instructions for a "PYTHON CALCULATOR FUNCTION WITH ERROR HANDLING". It asks the user to restore specific functions and implement them in `marks_utils.py`. Below this, there are sections for "Planned and implemented conversion program functions" and "Managed and tracked todo items for planning". A "Todos" section at the bottom indicates 1 file changed.