

AI Assisted Coding

Name : A.Anjali

Hall : 2303a52385

Batch : 33

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.

The screenshot shows the AI Assistant IDE interface. On the left, the code editor displays a Python script named assignment3.2.py. The code defines a calculator function that handles addition, subtraction, multiplication, and division. It includes error handling for non-numeric input and division by zero. The right side of the interface features a sidebar with a todo list and a progress bar. The todo list includes items like "Created todo list and implemented calculator function", "Managed todo items and built calculator function", and "Updated todo list and provided progress update". The progress bar shows the task is 77% complete. Below the code editor, the terminal window shows the command `python assignment3.2.py` being run, followed by user input for numbers and an operator, and the resulting output.

```
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:
        result = None

    print(f"Result: {result}")
```

```
PS C:\Users\95425\Downloads\AIAssistant> python -u "c:/Users/95425/Downloads/AIAssistant/assignment3.2.py"
Enter the first number: 15
Enter the second number: 5
Enter an operator (+ - * /): +
Result: 15.0 + 5.0 = 20
```

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.

```

def sort_marks_desc(marks):
    """ValueError if any mark is not an integer or out of 0..100 range.

    This uses Python's built-in 'sorted' (Timsort), which runs in O(n log n)
    and is efficient in practice.

    """
    if not isinstance(marks, (list, tuple)):
        raise TypeError("marks must be a list or tuple of integers 0..100")
    validated = []
    for m in marks:
        if not isinstance(m, int):
            raise ValueError(f"All marks must be integers between 0 and 100, got {m!r}")
        if m < 0 or m > 100:
            raise ValueError(f"All marks must be between 0 and 100, got {m!r}")
        validated.append(m)
    # Use built-in sorted (Timsort) for efficiency
    return sorted(validated, reverse=True)

if __name__ == '__main__':
    # Quick manual test
    print(sort_marks_desc([56, 90, 100, 8, 75]))

```

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.

```

def is_prime(n):
    """Handles edge cases: negatives, 0, and 1 are not prime.
    Efficiently checks divisibility using 6k +/- 1 optimization up to sqrt(n).

    """
    if not isinstance(n, int):
        raise TypeError("n must be an integer")
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

if __name__ == '__main__':
    for val in (2, 4, 1, -3, 17, 18):
        print(f"Input: {val} | Output: {'Prime' if is_prime(val) else 'Not Prime'}")

```

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

```

28
29 def grade_from_percentage(pct):
30     """Return grade letter for percentage using given mapping.
31
32     A: >=90, B: >=75, C: >=60, D: < 60
33
34     try:
35         pct = float(pct)
36     except Exception:
37         return 'Invalid'
38     if pct >= 90:
39         return 'A'
40     if pct >= 75:
41         return 'B'
42     if pct >= 60:
43         return 'C'
44     return 'D'
45
46
47 def grade_student(name, marks):
48     """Compute total, percentage, and grade for a student.
49
50     'marks' may be a dict mapping subject->mark or an iterable of numeric marks.
51     Returns a dict with 'name', 'marks' (as dict), 'total', 'percentage', 'grade'.
52     Percentage is calculated as the average mark (treating each subject as out of 100).
53
54     # Normalize marks into an ordered dict-like mapping
55     if isinstance(marks, dict):
56         subj_marks = dict(marks)
57     else:
58         # assume iterable of marks without subject names
59         subj_marks = {f'Subj {i+1}': float(m) for i, m in enumerate(marks)}
60
61     # 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

```

47 def grade_student(name, marks):
48     # Validate numeric marks and compute total
49     total = 0.0
50     count = 0
51     for k, v in list(subj_marks.items()):
52         try:
53             mv = float(v)
54         except Exception:
55             mv = 0.0
56         subj_marks[k] = mv
57         total += mv
58         count += 1
59
60     if count == 0:
61         percentage = 0.0
62     else:
63         average = total / count
64         percentage = average # treating average as percentage (out of 100)
65
66     grade = grade_from_percentage(percentage)
67     return {
68         'name': name,
69         'marks': subj_marks,
70         'total': total,
71         'percentage': round(percentage, 2),
72         'grade': grade,
73     }
74
75
76 def interactive_grading_system():
77     """Interactively accept multiple students and their subject marks.
78
79     For each student:
80
81     Enter student name and marks separated by commas.
82
83     Example:
84     Student 1: John, 85, 90, 78, 89, 92
85
86     Student 2: Jane, 72, 88, 95, 80, 98
87
88     Student 3: ...
89
90     Enter 'q' to quit.
91
92     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 -108-1`
- 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

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 at the bottom to handle keyboard interrupts and provide a summary result.

```
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 a main block to run the interactive system directly. The analysis pane continues to provide insights into the code's structure and potential improvements.

```
if __name__ == '__main__':
    # Run the interactive grading system when executed directly.
    interactive_grading_system()
```

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.

The screenshot shows a Python code editor with several files open:

- assignment3.py**: A script containing a unit conversion tool. It defines a function `convert_distance` that takes a value and a unit ('km' or 'miles') and returns the converted distance. It also includes a loop for user interaction.
- marks_utils.py**: A file containing utility functions for grading.
- grading_system.py**: A script containing a function `calculate_results` which calls `convert_distance`.
- conversions.py**: A file containing two conversion functions: `km_to_miles` and `miles_to_km`.

The right side of the interface displays a preview of the `conversions.py` code, showing its purpose as a calculator function with error handling. It also shows a task for writing a program that includes two conversion functions: `km_to_miles` and `miles_to_km`, with specific requirements for input and output.

The screenshot shows a Python code editor interface with the following details:

- File Explorer:** Shows files: assignment3.py, marks_utility.py, grading_system.py, and conversions.py.
- Code Editor:** The main pane displays the `grading_system.py` file content. The code defines a `convert_distance` function that takes a value and a unit, converts it from km to miles or vice versa, and returns the result. It also contains an `interactive_convert_loop` function that runs a conversion loop until the user exits.
- Output Panel:** Shows the command line interface (CLI) output for running the program. It includes error messages about missing implementations for `km_to_miles` and `miles_to_km`.
- Taskbar:** Includes tabs for "Assignment", "Python", "Grading System", and "Conversions".
- Help:** A sidebar on the right provides help for the `convert_distance` function, including its docstring and parameter descriptions.

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 also includes a try-except block to handle exceptions like `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 includes steps to restore specific files, implement functions like `km_to_miles` and `miles_to_km`, and run the program interactively.

Below the main code editor, there is a "Todos" section indicating 1 file changed.