

# AI Assisted Coding

Name :K.Srija

Hall : 2303a52189

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.

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 takes three parameters: `a`, `b`, and `op`. It handles numeric input, supports basic operations (+, -, \*, /), and handles division by zero. The right side of the interface shows a detailed progress log:

- Created todo list and implemented calculator function
- Managed todo items and built calculator function
- Called `apply_patch` with explanation string included
- Edited `assignment3.2.py`
- Updated todo list and provided progress update

The log also includes a warning about PowerShell compatibility and a terminal window showing the execution of the script and its output.

### 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. On the left, there's a file tree with files like assignment3.py, marks\_utils.py, and assignment2.py. The main editor window contains Python code for a sorting function:

```

6 def sort_marks_desc(marks):
    """ValueError if any mark is not an integer or out of 0..100 range.
    """
    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("All marks must be integers between 0 and 100, got {}")
        if m < 0 or m > 100:
            raise ValueError("All marks must be between 0 and 100, got {}")
        validated.append(m)
    # Use built-in sorted (Timsort) for efficiency
    return sorted(validated, reverse=True)

35 if __name__ == '__main__':
    # Quick manual test
    print(sort_marks_desc([56, 90, 100, 0, 75]))

```

The right panel shows a "CHAT" section with a message from the AI asking for a Python function to sort marks in descending order. Below it is a "TODO" list with items like "Planned function implementation" and "Created and updated task planning patch". At the bottom, there's a terminal window showing command-line interactions.

### 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. The file tree includes assignment2.py, marks\_utils.py, and marks\_utils2.py. The main editor window contains Python code for a prime number validation function:

```

5 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

27 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'}")

```

The right panel shows a "CHAT" section with a message from the AI asking for a prime number checker. Below it is a "TODO" list with items like "Implemented prime function and added is\_prime function" and "Managed and tracked todo items". At the bottom, there's a terminal window showing command-line interactions.

### 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 L
- 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.py` (200)  
How to run demo  
python < "import grading\_system; grading\_system.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.py` (200) — contains the grading CLI and functions (`calculate_results`, `assign_grade`, `display_results`, `check_scores`, `interactive_grading_system`).
- `marks_utils.py` : `marks_utils.py` — currently empty.

Shall I restore the `__init__.py` and `setup_wkns_demo` 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
85
86     Student 2: Jane, 72, 88, 95, 80
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 L
- 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.py` (200)  
How to run demo  
python < "import grading\_system; grading\_system.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.py` (200) — contains the grading CLI and functions (`calculate_results`, `assign_grade`, `display_results`, `check_scores`, `interactive_grading_system`).
- `marks_utils.py` : `marks_utils.py` — currently empty.

Shall I restore the `__init__.py` and `setup_wkns_demo` 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 file 'grading\_system.py' open. The code defines an interactive grading system. A tooltip from the AI Assistant is displayed, suggesting the addition of error handling and demo code. The code includes functions for calculating total, percentage, and grade, and a main loop for accepting student data.

```
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 -----")
        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 -----")
        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
```

The screenshot shows the AI Assistant interface with the file 'grading\_system.py' open. The code defines an interactive grading system. A tooltip from the AI Assistant is displayed, suggesting the addition of error handling and demo code. The code includes functions for calculating total, percentage, and grade, and a main loop for accepting student data.

```
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 = float(input(' Mark (0-100): '))
                subjects[subj] = mark

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

    except KeyboardInterrupt:
        print("\n----- Result -----")
        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 -----")
        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
```

The screenshot shows the AI Assistant interface with the file 'grading\_system.py' open. The code defines an interactive grading system. A tooltip from the AI Assistant is displayed, suggesting the addition of error handling and demo code. The code includes functions for calculating total, percentage, and grade, and a main loop for accepting student data.

```
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 = float(input(' Mark (0-100): '))
                subjects[subj] = mark

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

    except KeyboardInterrupt:
        print("\n----- Result -----")
        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 -----")
        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.

```

1  """Unit conversion tool
2
3  Defines 'convert_distance(value, unit)' and provides an interactive loop
4  to perform multiple conversions until the user exits.
5
6  Supported units: 'km' (kilometers) and 'miles'.
7  Conversion: 1 km = 0.621371 miles
8
9
10 def convert_distance(value, unit):
11     """Convert distance between kilometers and miles.
12
13     Parameters:
14         - value: numeric distance (int/float or string parseable to float)
15         - unit: string, either 'km' to convert km->miles or 'miles' to convert miles->km
16
17     Returns:
18         - converted_value (float rounded to 2 decimals), output_unit (str)
19
20     Raises:
21         - ValueError for invalid unit or non-numeric value
22
23     if unit is None:
24         raise ValueError('Unit must be specified as "km" or "miles"')
25     u = unit.strip().lower()
26     try:
27         v = float(value)
28     except Exception:
29         raise ValueError('Distance value must be numeric')
30
31     if u == 'km':
32         return (round(v * 0.621371, 2), 'miles')
33     if u == 'miles' or u == 'mile':
34         return (round(v / 0.621371, 2), 'km')
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

```

```

10 def convert_distance(value, unit):
11     u = unit.strip().lower()
12     try:
13         v = float(value)
14     except Exception:
15         raise ValueError('Distance value must be numeric')
16
17     if u == 'km':
18         return (round(v * 0.621371, 2), 'miles')
19     if u == 'miles' or u == 'mile':
20         return (round(v / 0.621371, 2), 'km')
21     raise ValueError(f'Invalid unit: {unit!r}. Use "km" or "miles"')
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

```

The screenshot shows a code editor interface with two main panes. The left pane displays a Python script named `grading_system.py`. The right pane shows a task titled "PYTHON CALCULATOR FUNCTION WITH ERROR HANDLING".

**Left Pane (Code Editor):**

```
File Edit Selection View Go Run Terminal Help < →
AIIAssistant
assignment12.py marks_utils.py grading_system.py conversions.py
grading_system.py
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()
```

**Right Pane (Task):**

PYTHON CALCULATOR FUNCTION WITH ERROR HANDLING

- grading\_system.py : grading\_system.py(209) — contains the grading CLI and functions calculate\_result, assign\_grade, display\_results, collect\_marks\_interactive, demo.
- marks\_utils.py : marks\_utils.py — currently empty.

Shell! restore the `sk_arctree` and `sent_marks_desc` implementations into `marks_utils.py` and paste the code here!

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(200)
- Functions: `km_to_miles(mile: float) -> float`, `miles_to_km(miles: float) -> float`, with accurate conversion factors and docstrings.
- Run interactively

python conversions.py

Programmatic use

```
new 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?

Tools (5/5)  
1 file changed +10 -1