AI ASSISTED CODING

2303A52490

ASSIGNMENT 3

Task 1: Progressive Prompting for Calculator Design

```python
def calculator():
    num1 = float(input("Enter first number: "))
    operator = input("Enter operator (+, -, *, /): ")
    num2 = float(input("Enter second number: "))

    if operator == '+':
        result = num1 + num2
    elif operator == '-':
        result = num1 - num2
    elif operator == '*':
        result = num1 * num2
    elif operator == '/':
        if num2 != 0:
            result = num1 / num2
        else:
            print("Error: Division by zero")
            return
    else:
        print("Invalid operator")
        return

    print(f"Result: {result}")

calculator()
```

after giving detailed prompt

```python
import math

class Calculator:
    def __init__(self):
        self.memory = 0

    def add(self, x, y):
        return x + y

    def subtract(self, x, y):
        return x - y

    def multiply(self, x, y):
        return x * y

    def divide(self, x, y):
        try:
            return x / y
        except ZeroDivisionError:
            print("Error: Cannot divide by zero")
            return None

    def power(self, x, y):
        return math.pow(x, y)

    def square_root(self, x):
        if x < 0:
            print("Error: Cannot calculate square root of negative number")
            return None
        return math.sqrt(x)

    def store_in_memory(self, value):
        self.memory = value
        print(f"Value {value} stored in memory")
```

```
    def recall_memory(self):
        print(f"Recalled value: {self.memory}")
        return self.memory

    def show_menu(self):
        print("\n===== Advanced Calculator =====")
        print("1. Addition (+)")
        print("2. Subtraction (-)")
        print("3. Multiplication (*)")
        print("4. Division (/)")
        print("5. Power (x^y)")
        print("6. Square Root (√x)")
        print("7. Store result in memory")
        print("8. Recall from memory")
        print("9. Exit")
        print("============================")

    def get_number_input(self, prompt):
        while True:
            try:
                return float(input(prompt))
            except ValueError:
                print("Error: Please enter a valid number")

    def calculator(self):
        print("Welcome to the Advanced Calculator!")

        while True:
            self.show_menu()
            choice = input("Enter your choice (1-9): ")

            if choice == '9':
                print("Thank you for using the Advanced Calculator!")
                break

            if choice in ['1', '2', '3', '4', '5']:
                num1 = self.get_number_input("Enter first number: ")
                num2 = self.get_number_input("Enter second number: ")
```

```
            if choice == '1':
                result = self.add(num1, num2)
                print(f"{num1} + {num2} = {result}")
            elif choice == '2':
                result = self.subtract(num1, num2)
                print(f"{num1} - {num2} = {result}")
            elif choice == '3':
                result = self.multiply(num1, num2)
                print(f"{num1} * {num2} = {result}")
            elif choice == '4':
                result = self.divide(num1, num2)
                if result is not None:
                    print(f"{num1} / {num2} = {result}")
            elif choice == '5':
                result = self.power(num1, num2)
                print(f"{num1} ^ {num2} = {result}")

            if result is not None:
                store = input("Store this result in memory? (y/n): ").lower()
                if store == 'y':
                    self.store_in_memory(result)

        elif choice == '6':
            num = self.get_number_input("Enter number for square root: ")
            result = self.square_root(num)
            if result is not None:
                print(f"√{num} = {result}")
                store = input("Store this result in memory? (y/n): ").lower()
                if store == 'y':
                    self.store_in_memory(result)

        elif choice == '7':
            value = self.get_number_input("Enter value to store in memory: ")
            self.store_in_memory(value)

        elif choice == '8':
            self.recall_memory()

        else:
            print("Invalid choice. Please enter a number between 1 and 9.")

        input("\nPress Enter to continue...")

__name__ == "__main__":
calc = Calculator()
calc.calculator()
```

output

```
Welcome to the Advanced Calculator!

===== Advanced Calculator =====
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Power (x^y)
6. Square Root (√x)
7. Store result in memory
8. Recall from memory
9. Exit
============================
Enter your choice (1-9): 1
Enter first number: 10
Enter second number: 5
10.0 + 5.0 = 15.0
Store this result in memory? (y/n): y
Value 15.0 stored in memory

Press Enter to continue...
```

Task 2: Refining Prompts for Sorting Logic

```python
def sort_marks(marks):
    return sorted(marks)

student_marks = [85, 92, 78, 90, 88]
sorted_marks = sort_marks(student_marks)
print(sorted_marks)
```

after giving detailed prompt

```python
def sort_student_marks(students, sort_by="marks", order="desc", min_marks=None):
    if sort_by not in ["marks", "name"]:
        raise ValueError("sort_by parameter must be 'marks' or 'name'")

    if order not in ["asc", "desc"]:
        raise ValueError("order parameter must be 'asc' or 'desc'")

    filtered_students = students
    if min_marks is not None:
        filtered_students = [s for s in students if s.get("marks", 0) >= min_marks]
        if not filtered_students:
            print(f"Warning: No students found with marks >= {min_marks}")

    valid_students = []
    for student in filtered_students:
        if not isinstance(student, dict):
            print(f"Warning: Skipping invalid student record (not a dictionary): {student}")
            continue

        if "name" not in student or "marks" not in student:
            print(f"Warning: Skipping student with missing required fields: {student}")
            continue

        valid_students.append(student)

    reverse = (order == "desc")

    if sort_by == "marks":
        sorted_students = sorted(valid_students, key=lambda x: x["marks"], reverse=reverse)
    else:
        sorted_students = sorted(valid_students, key=lambda x: x["name"].lower(), reverse=reverse)

    return sorted_students

students = [
    {"name": "Alice", "marks": 85},
    {"name": "Bob", "marks": 92},
    {"name": "Charlie", "marks": 78},
    {"name": "David", "marks": 88},
    {"name": "Eva", "marks": 92},
    {"name": "Frank", "marks": 72}
]

print("\nTop students (sorted by marks, descending):")
top_students = sort_student_marks(students, sort_by="marks", order="desc")
for student in top_students:
    print(f"{student['name']}: {student['marks']}")

print("\nStudents sorted by name (ascending):")
sorted_by_name = sort_student_marks(students, sort_by="name", order="asc")
for student in sorted_by_name:
    print(f"{student['name']}: {student['marks']}")

print("\nHigh-performing students (marks >= 85):")
high_performers = sort_student_marks(students, sort_by="marks", order="desc", min_marks=85)
for student in high_performers:
    print(f"{student['name']}: {student['marks']}")

students_with_invalid = students.copy()
students_with_invalid.append({"name": "Invalid Student"})
students_with_invalid.append("Not a dictionary")

try:
    result = sort_student_marks(students_with_invalid, sort_by="marks", order="desc")
    print("\nSorted valid students after filtering invalid entries:")
    for student in result:
        print(f"{student['name']}: {student['marks']}")
except Exception as e:
    print(f"Error: {e}")
```

output

```
Top students (sorted by marks, descending):
Bob: 92
Eva: 92
David: 88
Alice: 85
Charlie: 78
Frank: 72
```

Task 3: Few-Shot Prompting for Prime Number Validation

```python
def is_prime(n):
    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

test_cases = [0, 1, 2, 3, 4, 17, 25, 29, 97, 100, -5]

print("Testing is_prime function:")
print("-" * 40)
for num in test_cases:
    result = is_prime(num)
    print(f"is_prime({num}) = {result}")

large_prime = 999983
print(f"\nis_prime({large_prime}) = {is_prime(large_prime)}")

non_prime = 1000000
print(f"is_prime({non_prime}) = {is_prime(non_prime)}")
```

output

```
Testing is_prime function:
----------------------------------------
is_prime(0) = False
is_prime(1) = False
is_prime(2) = True
is_prime(3) = True
is_prime(4) = False
is_prime(17) = True
is_prime(25) = False
is_prime(29) = True
is_prime(97) = True
is_prime(100) = False
is_prime(-5) = False

is_prime(999983) = True
is_prime(1000000) = False
```

Task 4: Prompt-Guided UI Design for Student Grading System

```
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
import datetime

class StudentGradingSystem:
    def __init__(self, root):
        self.root = root
        self.root.title("Student Grading System")
        self.root.geometry("800x600")
        self.root.configure(bg="#f0f0f0")

        main_frame = ttk.Frame(root, padding="20")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        info_frame = ttk.LabelFrame(main_frame, text="Student Information",
padding="10")
        info_frame.grid(row=0, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=10)

        ttk.Label(info_frame, text="Student Name:").grid(row=0, column=0, sticky=tk.W,
pady=5)
        self.name_var = tk.StringVar()
        ttk.Entry(info_frame, textvariable=self.name_var, width=30).grid(row=0, column=1,
sticky=tk.W, pady=5, padx=5)

        ttk.Label(info_frame, text="Student ID:").grid(row=1, column=0, sticky=tk.W,
pady=5)
        self.id_var = tk.StringVar()
```

```
    ttk.Entry(info_frame, textvariable=self.id_var, width=30).grid(row=1, column=1, sticky=tk.W, pady=5, padx=5)

    marks_frame = ttk.LabelFrame(main_frame, text="Subject Marks (0-100)", padding="10")
    marks_frame.grid(row=1, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=10)

    subjects = ["Mathematics", "Science", "English", "History", "Computer Science"]
    self.mark_vars = {}

    for i, subject in enumerate(subjects):
        ttk.Label(marks_frame, text=f"{subject}:").grid(row=i, column=0, sticky=tk.W, pady=5)
        self.mark_vars[subject] = tk.StringVar(value="0")
        ttk.Entry(marks_frame, textvariable=self.mark_vars[subject], width=10).grid(row=i, column=1, sticky=tk.W, pady=5, padx=5)

    button_frame = ttk.Frame(main_frame)
    button_frame.grid(row=2, column=0, columnspan=2, pady=15)

    ttk.Button(button_frame, text="Calculate Grade", command=self.calculate_grade).grid(row=0, column=0, padx=10)
    ttk.Button(button_frame, text="Clear Form", command=self.clear_form).grid(row=0, column=1, padx=10)
    ttk.Button(button_frame, text="Save Results", command=self.save_results).grid(row=0, column=2, padx=10)
    ttk.Button(button_frame, text="Help", command=self.show_help).grid(row=0, column=3, padx=10)

    result_frame = ttk.LabelFrame(main_frame, text="Results", padding="10")
```

```
        result_frame.grid(row=3, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=10)

        self.result_text = scrolledtext.ScrolledText(result_frame, width=70, height=8, font=("Arial", 10))
        self.result_text.grid(row=0, column=0, sticky=(tk.W, tk.E))
        self.result_text.configure(state='disabled')

        self.status_var = tk.StringVar()
        self.status_var.set("Ready to calculate grade")
        status_bar = ttk.Label(root, textvariable=self.status_var, relief=tk.SUNKEN, anchor=tk.W)
        status_bar.grid(row=1, column=0, sticky=(tk.W, tk.E))

    def validate_marks(self):
        for subject, var in self.mark_vars.items():
            try:
                mark = float(var.get())
                if mark < 0 or mark > 100:
                    messagebox.showerror("Input Error", f"{subject} marks must be between 0 and 100")
                    return False
            except ValueError:
                messagebox.showerror("Input Error", f"Please enter a valid number for {subject} marks")
                return False
        return True

    def calculate_grade(self):
        if not self.validate_marks() or not self.name_var.get().strip():
```

```
    return

marks = [float(var.get()) for var in self.mark_vars.values()]
total = sum(marks)
percentage = (total / 500) * 100

if percentage >= 90:
    grade = "A+"
    grade_color = "green"
elif percentage >= 80:
    grade = "A"
    grade_color = "green"
elif percentage >= 70:
    grade = "B"
    grade_color = "blue"
elif percentage >= 60:
    grade = "C"
    grade_color = "blue"
elif percentage >= 50:
    grade = "D"
    grade_color = "orange"
else:
    grade = "F"
    grade_color = "red"

result = f"Student Name: {self.name_var.get()}\n"
result += f"Student ID: {self.id_var.get()}\n"
result += f"{'='*40}\n"
```

```
        for subject, mark in zip(self.mark_vars.keys(), marks):
            result += f"{subject}: {mark}/100\n"
        result += f"{'='*40}\n"
        result += f"Total Marks: {total}/500\n"
        result += f"Percentage: {percentage:.2f}%\n"
        result += f"Grade: {grade}\n"

        self.result_text.configure(state='normal')
        self.result_text.delete(1.0, tk.END)
        self.result_text.insert(tk.END, result)

        grade_start = result.find(f"Grade: {grade}") + 7
        grade_end = grade_start + len(grade)
        self.result_text.tag_add("grade", f"1.{grade_start}", f"1.{grade_end}")
        self.result_text.tag_config("grade", foreground=grade_color, font=("Arial", 10, "bold"))

        self.result_text.configure(state='disabled')
        self.status_var.set(f"Grade calculated successfully! Grade: {grade}, Percentage: {percentage:.2f}%")

        self.current_result = {
            "name": self.name_var.get(),
            "id": self.id_var.get(),
            "marks": dict(zip(self.mark_vars.keys(), marks)),
            "total": total,
            "percentage": percentage,
            "grade": grade
        }
```

```
    def clear_form(self):
        self.name_var.set("")
        self.id_var.set("")
        for var in self.mark_vars.values():
            var.set("0")

        self.result_text.configure(state='normal')
        self.result_text.delete(1.0, tk.END)
        self.result_text.configure(state='disabled')

        self.status_var.set("Form cleared. Ready for new entry.")

    def save_results(self):
        if not hasattr(self, 'current_result') or not self.current_result:
            messagebox.showinfo("No Results", "Please calculate grades first before saving.")
            return

        try:
            filename = f"grade_report_{self.current_result['name'].replace(' ',
'_')}_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.txt"
            with open(filename, 'w') as f:
                f.write("STUDENT GRADE REPORT\n")
                f.write("=" * 40 + "\n")
                f.write(f"Name: {self.current_result['name']}\n")
                f.write(f"ID: {self.current_result['id']}\n")
                f.write(f"Date: {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
                f.write("-" * 40 + "\n")
```

```
            f.write("SUBJECT MARKS:\n")
            for subject, mark in self.current_result['marks'].items():
                f.write(f"{subject}: {mark}/100\n")
            f.write("-" * 40 + "\n")
            f.write(f"Total Marks: {self.current_result['total']}/500\n")
            f.write(f"Percentage: {self.current_result['percentage']:.2f}%\n")
            f.write(f"Grade: {self.current_result['grade']}\n")
            f.write("=" * 40 + "\n")
            f.write("This is an auto-generated report from Student Grading System")

        self.status_var.set(f"Results saved successfully to {filename}")
        messagebox.showinfo("Save Successful", f"Grade report saved to {filename}")
    except Exception as e:
        messagebox.showerror("Save Error", f"Failed to save results: {str(e)}")

def show_help(self):
    help_text = """
Student Grading System Help

1. Enter the student's name and ID in the respective fields.
2. Enter marks for each subject (0-100) in the subject fields.
3. Click "Calculate Grade" to compute the total marks, percentage, and grade.
4. Results will be displayed in the Results section with color-coded grades.
5. Click "Save Results" to save the current results to a text file.
6. Click "Clear Form" to reset all fields for a new student.

Grading Scale:
- A+ : 90-100%
```

- A : 80-89%

- B : 70-79%

- C : 60-69%

- D : 50-59%

- F : Below 50%

Note: All marks must be between 0 and 100. The system will validate your inputs.

"""

messagebox.showinfo("Help", help_text)

```
if __name__ == "__main__":
    root = tk.Tk()
    app = StudentGradingSystem(root)
    root.mainloop()
```

Task 5: Analyzing Prompt Specificity in Unit Conversion Functions

```
def convert_distance(value, from_unit, to_unit):
    from_unit = from_unit.lower()
    to_unit = to_unit.lower()

    if value < 0:
        raise ValueError("Distance value cannot be negative")

    supported_units = ["km", "kilometer", "kilometers", "miles", "mile", "m", "meter",
"meters", "ft", "foot", "feet"]

    unit_map = {
```

```
        "km": "km", "kilometer": "km", "kilometers": "km",
        "miles": "miles", "mile": "miles",
        "m": "m", "meter": "m", "meters": "m",
        "ft": "ft", "foot": "ft", "feet": "ft"
    }

    if from_unit not in unit_map:
        raise ValueError(f"Unsupported source unit: {from_unit}. Supported units are: km, miles, m, ft")
    if to_unit not in unit_map:
        raise ValueError(f"Unsupported target unit: {to_unit}. Supported units are: km, miles, m, ft")

    from_unit = unit_map[from_unit]
    to_unit = unit_map[to_unit]

    if from_unit == to_unit:
        return round(value, 4)

    KM_TO_MILES = 0.621371
    MILES_TO_KM = 1.60934
    KM_TO_METERS = 1000.0
    MILES_TO_FEET = 5280.0
    METERS_TO_KM = 0.001
    FEET_TO_MILES = 1 / 5280.0

    if from_unit == "km" and to_unit == "miles":
        result = value * KM_TO_MILES
    elif from_unit == "miles" and to_unit == "km":
```

```
    result = value * MILES_TO_KM
elif from_unit == "km" and to_unit == "m":
    result = value * KM_TO_METERS
elif from_unit == "m" and to_unit == "km":
    result = value * METERS_TO_KM
elif from_unit == "miles" and to_unit == "ft":
    result = value * MILES_TO_FEET
elif from_unit == "ft" and to_unit == "miles":
    result = value * FEET_TO_MILES
elif from_unit == "km" and to_unit == "ft":
    meters = value * KM_TO_METERS
    result = meters * 3.28084
elif from_unit == "ft" and to_unit == "km":
    meters = value / 3.28084
    result = meters * METERS_TO_KM
elif from_unit == "miles" and to_unit == "m":
    km = value * MILES_TO_KM
    result = km * KM_TO_METERS
elif from_unit == "m" and to_unit == "miles":
    km = value * METERS_TO_KM
    result = km * KM_TO_MILES
elif from_unit == "m" and to_unit == "ft":
    result = value * 3.28084
elif from_unit == "ft" and to_unit == "m":
    result = value * 0.3048
else:
    raise ValueError(f"Conversion from {from_unit} to {to_unit} is not supported")
```

```
    return round(result, 4)

def show_conversion_examples():
    examples = [
        (10, "km", "miles"),
        (5, "miles", "km"),
        (1, "km", "m"),
        (1, "miles", "ft"),
        (1000, "m", "km"),
        (5280, "ft", "miles"),
        (2.5, "m", "ft"),
        (100, "ft", "m"),
        (3, "km", "ft"),
        (10000, "ft", "km")
    ]

    print("Unit Conversion Examples:")
    print("-" * 50)
    for value, from_unit, to_unit in examples:
        try:
            result = convert_distance(value, from_unit, to_unit)
            print(f"{value} {from_unit} = {result} {to_unit}")
        except ValueError as e:
            print(f"Error converting {value} {from_unit} to {to_unit}: {e}")

show_conversion_examples()

print("\n" + "="*50)
```

```
print("Interactive Unit Conversion")
print("="*50)

while True:
    try:
        value = float(input("\nEnter value to convert (or 0 to exit): "))
        if value == 0:
            break

        from_unit = input("Convert from (km/miles/m/ft): ").lower()
        to_unit = input("Convert to (km/miles/m/ft): ").lower()

        result = convert_distance(value, from_unit, to_unit)
        print(f"\nResult: {value} {from_unit} = {result} {to_unit}")

    except ValueError as e:
        print(f"Error: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

print("\nThank you for using the Unit Conversion Tool!")
```

output