

## AI Assistant Coding Assignment-2

---

Name of Student : V.Shiva krishna      Batch : 41  
Enrollment No. : 2303A52151

---

### Task 1: Book Class Generation

Use Cursor AI to generate a Python class **Book** with attributes **title**, **author**, and a **summary()** method.

#### Prompt:

Generate a Python class named **Book** with attributes **title**, **author**, and a method **summary()** that returns book details.

#### Code:

```
# Task 1: Generate a Python class named Book with attributes title, author, and a method summary() that returns book details.
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
    def summary(self):
        return f"Title: {self.title}, Author: {self.author}"

if __name__ == "__main__":
    book1 = Book("Atomic Habits", "James Clear")
    print(book1.summary())
```

#### Output:

```
===== Task 1 Output =====
Title: Atomic Habits, Author: James Clear
```

## Justification:

Generated code using inline prompt of GitHub Copilot in VS Code for creating a Python class named Book with attributes title and author. The logic demonstrates object-oriented programming by defining a constructor to initialize data members and a method summary() to return formatted book details. An object is created in the main block, and the method is invoked to display the book information, showcasing class creation, object instantiation, and method usage in Python.

## Task 2: Sorting Dictionaries with AI

Use Gemini and Cursor AI to generate code that sorts a list of dictionaries by a key.

### Prompt:

*Sort user records by age.*

### Code:

```
# Task 2: Sort user records by age.
print("\n==== Task 2 Output =====")
user_records = [
    {"name": "John", "age": 25, "sub_id": 1, "subject": "English", "score": 90, "passed": True, "final_grade": "A", "course_id": 1},
    {"name": "Jane", "age": 22, "sub_id": 2, "subject": "Math", "score": 85, "passed": False, "final_grade": "B", "course_id": 2},
    {"name": "Jim", "age": 28, "sub_id": 3, "subject": "Science", "score": 88, "passed": True, "final_grade": "A", "course_id": 3},
    {"name": "Jill", "age": 21, "sub_id": 4, "subject": "History", "score": 92, "passed": False, "final_grade": "C", "course_id": 4},
]

sorted_records = sorted(user_records, key=lambda x: x["course_id"])
for record in sorted_records:
    print(record)
```

### Output:

```
==== Task 2 Output =====
{'name': 'John', 'age': 25, 'sub_id': 1, 'subject': 'English', 'score': 90, 'passed': True, 'final_grade': 'A', 'course_id': 1}
{'name': 'Jane', 'age': 22, 'sub_id': 2, 'subject': 'Math', 'score': 85, 'passed': False, 'final_grade': 'B', 'course_id': 2}
{'name': 'Jim', 'age': 28, 'sub_id': 3, 'subject': 'Science', 'score': 88, 'passed': True, 'final_grade': 'A', 'course_id': 3}
{'name': 'Jill', 'age': 21, 'sub_id': 4, 'subject': 'History', 'score': 92, 'passed': False, 'final_grade': 'C', 'course_id': 4}
```

### Justification:

Generated code using inline prompt of GitHub Copilot in VS Code to sort user records stored as a list of dictionaries. The logic uses Python's built-in `sorted()` function with a lambda expression as the key to arrange records based on `course_id`. This approach efficiently organizes structured data without modifying the original list, demonstrating the use of higher-order functions, lambda expressions, and dictionary-based data handling in Python. The sorted records are then iterated and printed in an ordered manner.

### Task 3: Calculator Using Functions

Ask Gemini to generate a calculator using functions and explain how it works.

### Prompt:

add a function to review a basic calculator module using functions and explain how it works. #add choosing options for the user to choose from the calculator functions. print the result of the chosen function and print the result of the chosen function if i select add they only add print add only dont want other functions to be printed.

### Code:

```
# Task 3: add a function to review a basic calculator module using functions and explain how it works. #add choosing options for the user to choose from the calculator functions

print("\n===== Task 3 Output =====")

# Calculator functions
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b
```

```
def divide(a, b):
    return a / b

def modulus(a, b):
    return a % b

def power(a, b):
    return a ** b

def square(a):
    return a ** 2

def cube(a):
    return a ** 3

def square_root(a):
    return math.sqrt(a)

# Menu
print("Choose a function:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")
print("5. Modulus")
print("6. Power")
print("7. Square")
print("8. Cube")
print("9. Square Root")

choice = int(input("Enter your choice: "))
```

## Output:

```
===== Task 3 Output =====
Choose a function:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulus
6. Power
7. Square
8. Cube
9. Square Root

Enter your choice: 1
Enter first number: 2
Enter second number: 3
Result: 5.0
```

## Justification:

Generated code using inline prompt of GitHub Copilot in VS Code to review and implement a basic calculator module using functions. The logic defines separate functions for each arithmetic operation such as addition, subtraction, multiplication, division, modulus, power, square, cube, and square root. A menu-driven approach is used to allow the user to choose a specific operation, and conditional statements ensure that only the selected function is executed and displayed. User inputs are taken dynamically based on the chosen operation, demonstrating modular programming, code reusability, and clear separation of logic. This implementation improves readability, maintainability, and user interaction in a calculator application.

**Task 4: # *Armstrong Number Optimization* Generate an Armstrong number**

*program using Gemini, then improve it using Cursor AI.*

## Prompt:

**Generate an Armstrong number program using Gemini, then improve it using Cursor AI.**

**Scenario: An existing solution is inefficient.**

## Code:

### Procedural:

```
# Task 4: Armstrong Number Optimization Generate an Armstrong number program using Gemini, then improve it using Cursor AI.  
print("\n==== Task 4 Output =====")  
def is_armstrong(number):  
    return sum(int(digit) ** len(str(number)) for digit in str(number)) == number  
  
number = int(input("Enter a number: "))  
if is_armstrong(number):  
    print(f"{number} is an Armstrong number")  
else:  
    print(f"{number} is not an Armstrong number")
```

## Output:

```
==== Task 4 Output =====  
Enter a number: 153  
153 is an Armstrong number
```

### **Justification:**

**Generated an initial Armstrong number program using Gemini with basic looping and arithmetic operations to check whether a number satisfies the Armstrong condition. The code was then optimized using Cursor AI by reducing redundant calculations, improving variable usage, and enhancing readability. The optimized version minimizes computational overhead and follows cleaner logic, resulting in better performance and maintainability compared to the original implementation.**

