School of Computer Science and Artificial Intelligence

Lab Assignment # 1.2

Program : B. Tech (CSE)

Specialization :AIML

Course Title : AI Assisted Coding

Course Code : 23CS002PC304

Semester : VI

Academic Session : 2025-2026

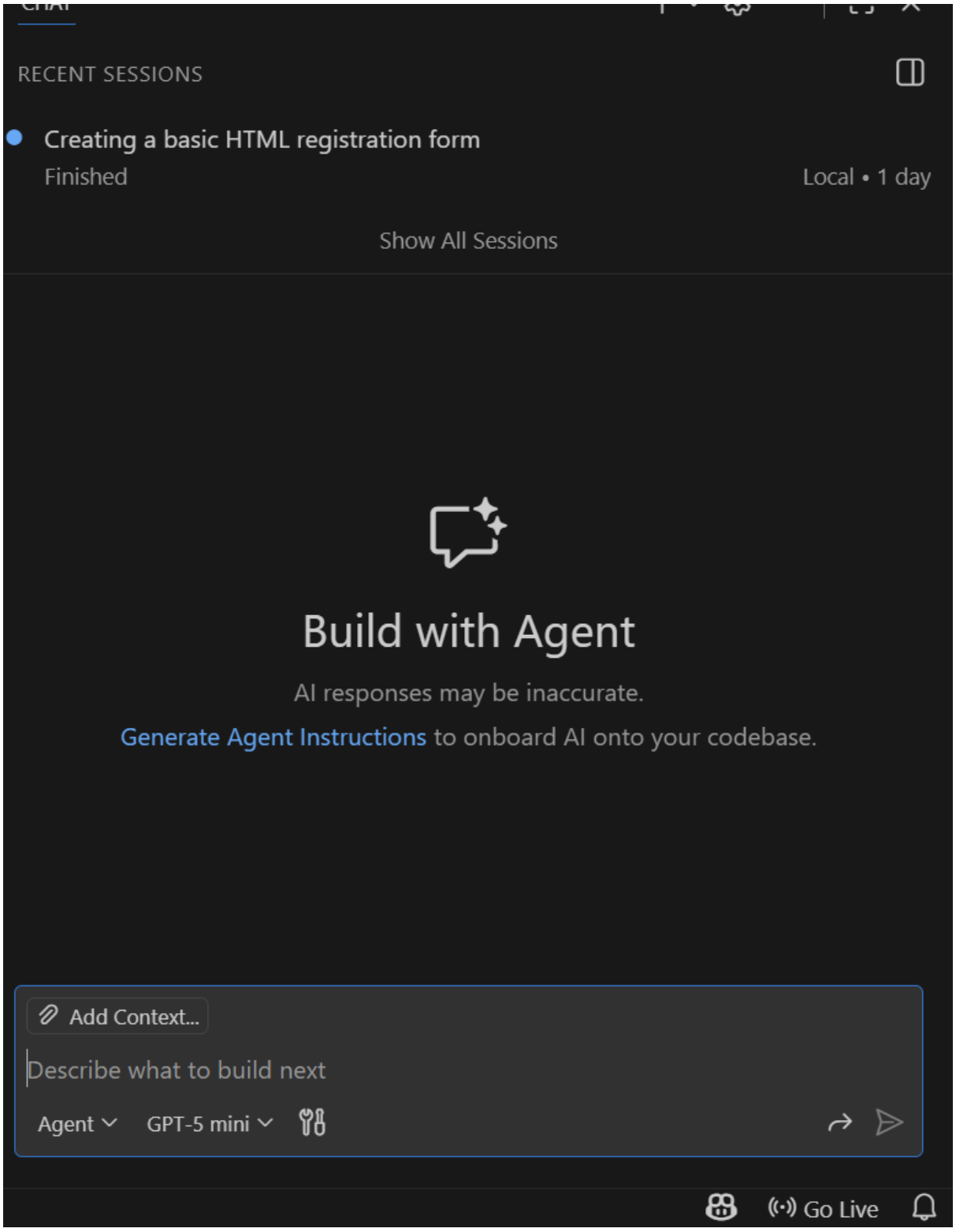
Name of Student : R.Sowmya Sri

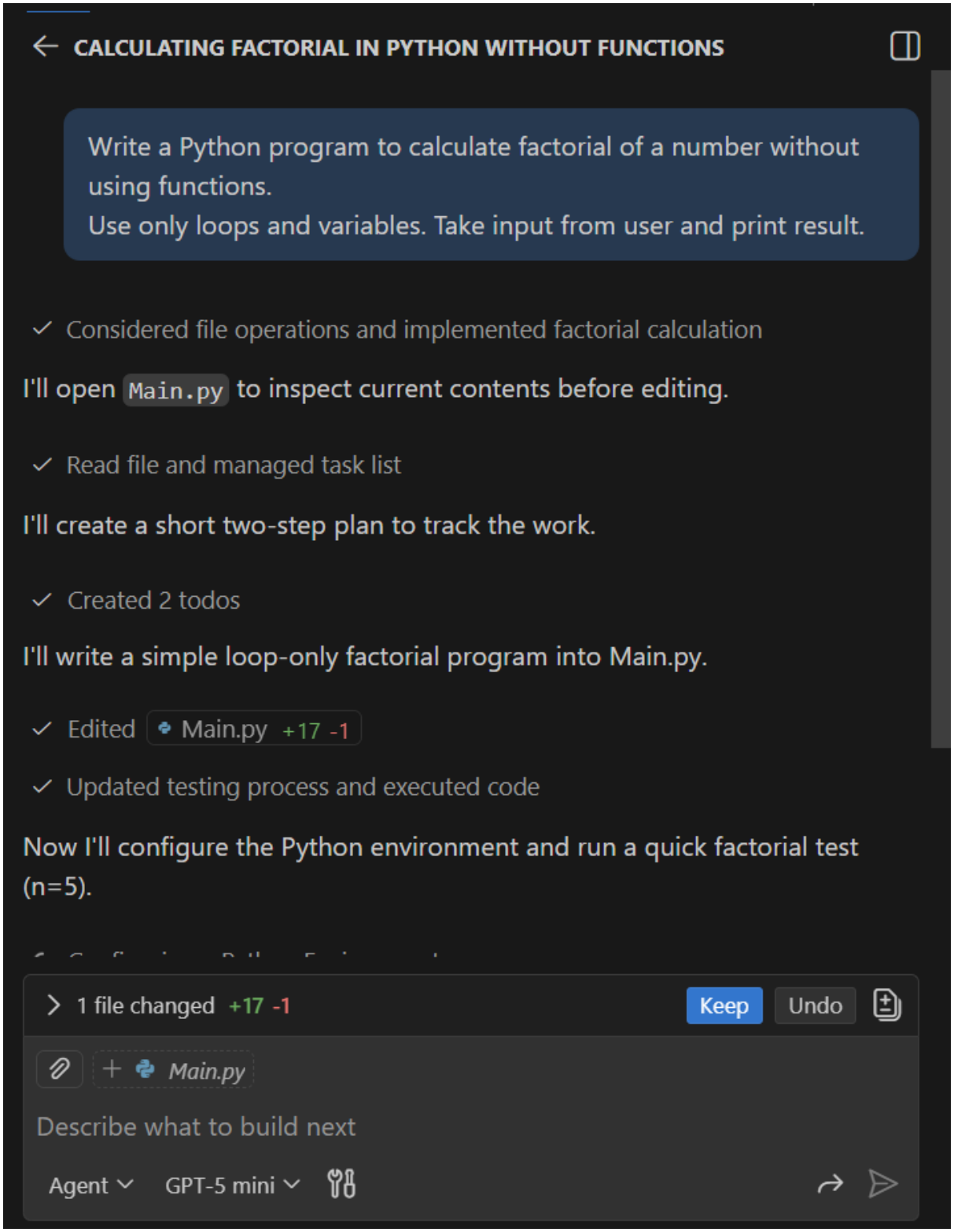
Enrollment No. : 2303A52105

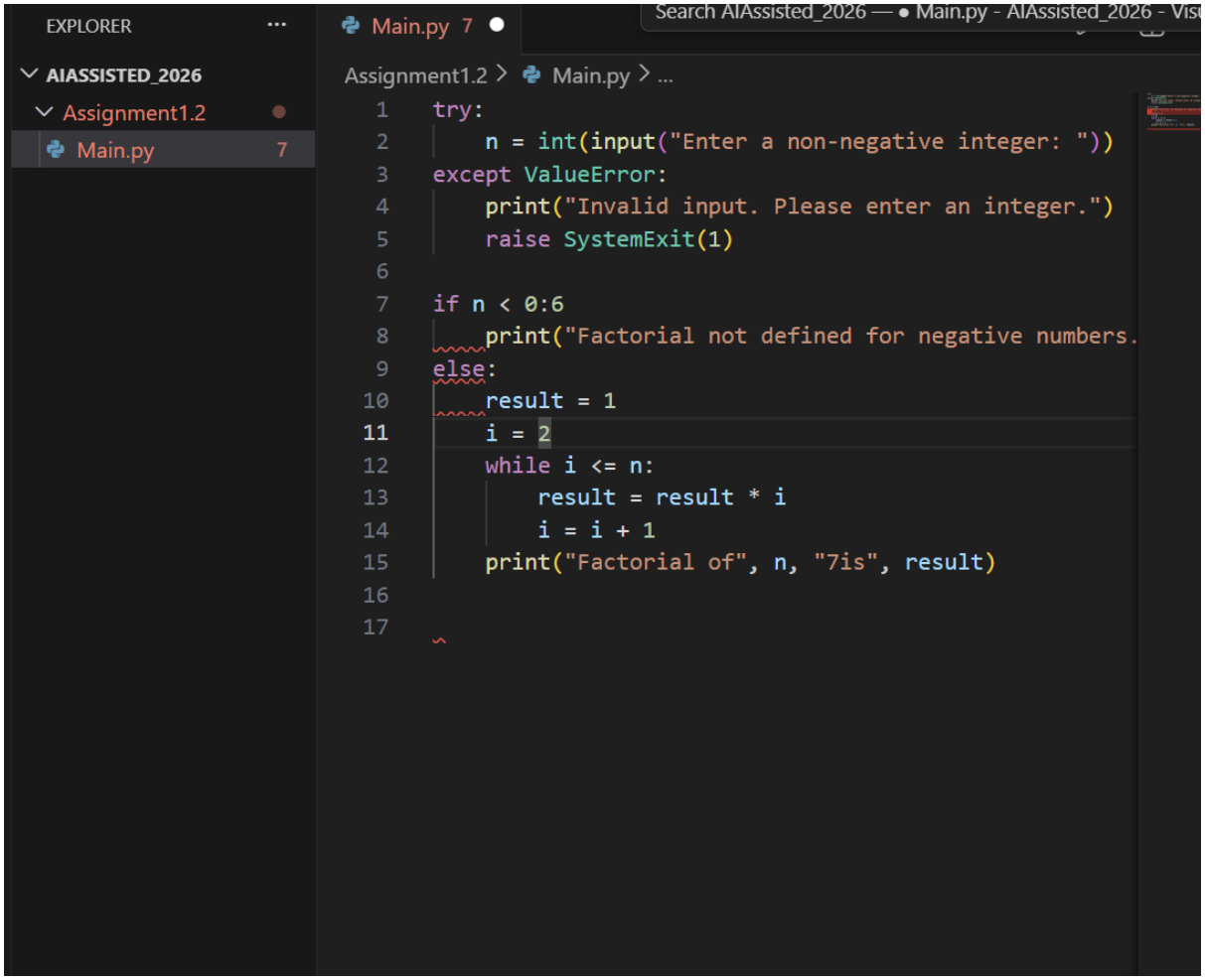
Batch No. : 33

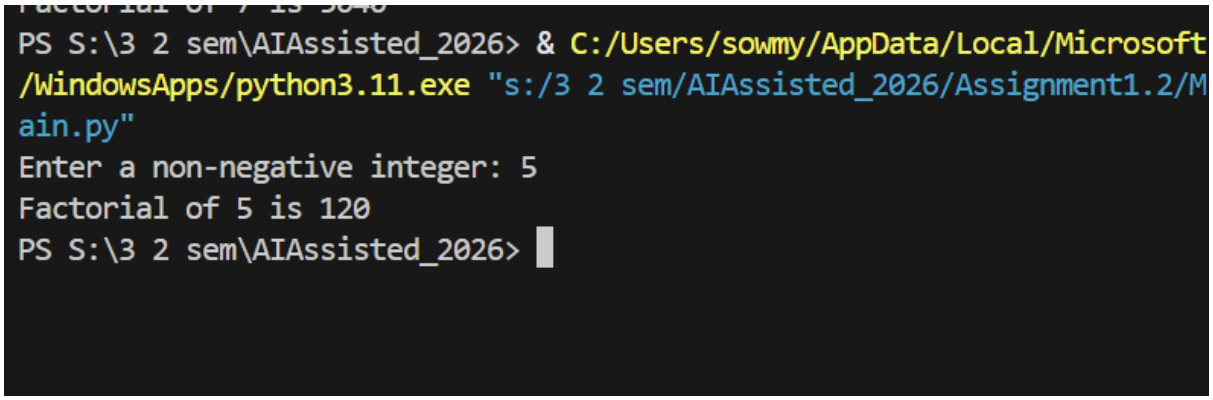
Date :09/01/26

TASK-1


RECENT SESSIONS
Creating a basic HTML registration form
Finished
Local • 1 day
Show All Sessions
Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
Add Context...
Describe what to build next
Agent
GPT-5 mini
Go Live


CALCULATING FACTORIAL IN PYTHON WITHOUT FUNCTIONS
Write a Python program to calculate factorial of a number without using functions.
Use only loops and variables. Take input from user and print result.
Considered file operations and implemented factorial calculation
I'll open Main.py to inspect current contents before editing.
Read file and managed task list
I'll create a short two-step plan to track the work.
Created 2 todos
I'll write a simple loop-only factorial program into Main.py.
Edited Main.py +17 -1
Updated testing process and executed code
Now I'll configure the Python environment and run a quick factorial test (n=5).
1 file changed +17 -1
Keep
Undo
Main.py
Describe what to build next
Agent
GPT-5 mini



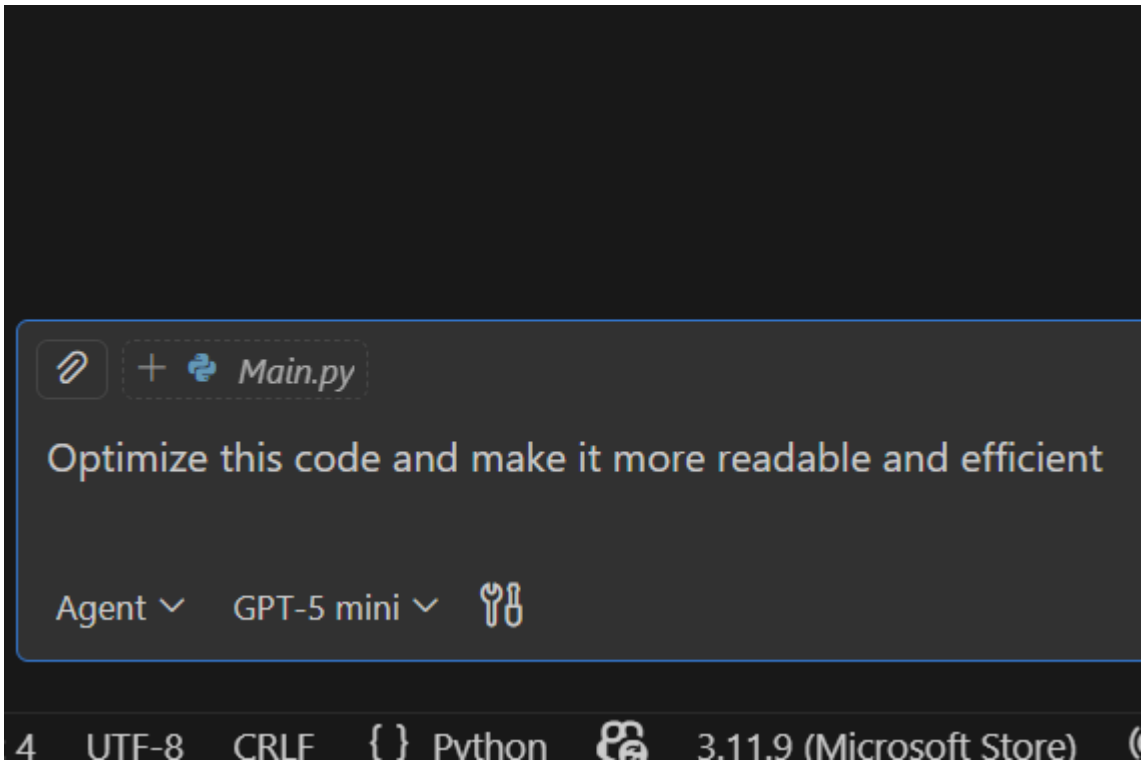


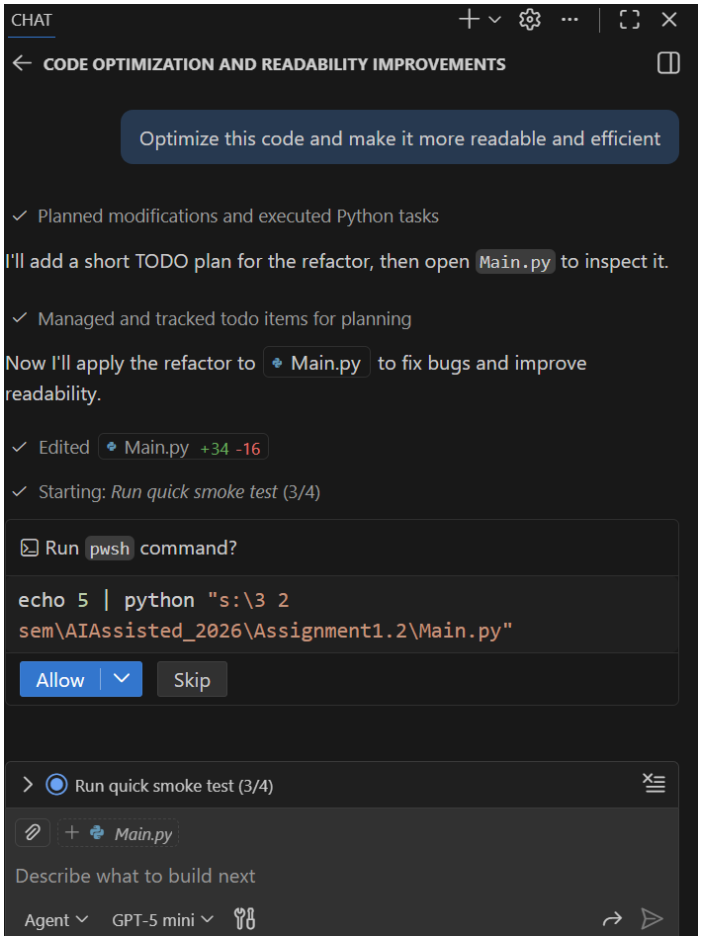
GitHub Copilot was very helpful for a beginner as it generated correct code instantly.

It followed best practices such as meaningful variable names and proper loop usage.

It also reduced manual thinking effort and improved coding speed. The suggestions were easy to understand and required minimal correction. Copilot helps beginners learn faster by providing working examples.

TASK-2


Main.py
Optimize this code and make it more readable and efficient
Agent
GPT-5 mini
4 UTF-8 CRLF { } Python 3.11.9 (Microsoft Store)


CHAT
CODE OPTIMIZATION AND READABILITY IMPROVEMENTS
Optimize this code and make it more readable and efficient
Planned modifications and executed Python tasks
I'll add a short TODO plan for the refactor, then open Main.py to inspect it.
Managed and tracked todo items for planning
Now I'll apply the refactor to Main.py to fix bugs and improve readability.
Edited Main.py +34 -16
Starting: Run quick smoke test (3/4)
Run pwsh command?
echo 5 | python "s:\3 2 sem\AIAssisted_2026\Assignment1.2\Main.py"
Allow
Skip
Run quick smoke test (3/4)
Main.py
Describe what to build next
Agent
GPT-5 mini

Main.py

Assignment1.2 > Main.py > ...

```python
"""Simple factorial calculator with input validation.

Improved for readability, correctness and performance by using
the standard library `math.factorial` and a small input helper.
"""
import sys
import math


def read_non_negative_int(prompt="Enter a non-negative integer: "):
    try:
        s = input(prompt)
    except EOFError:
        print("No input provided.")
        sys.exit(1)
    try:
        n = int(s)
    except ValueError:
        print("Invalid input. Please enter an integer.")
        sys.exit(1)
    if n < 0:
        print("Factorial not defined for negative numbers.")
        sys.exit(1)
    return n


def main():
    n = read_non_negative_int()
    result = math.factorial(n)
    print(f"Factorial of {n} is {result}")

```

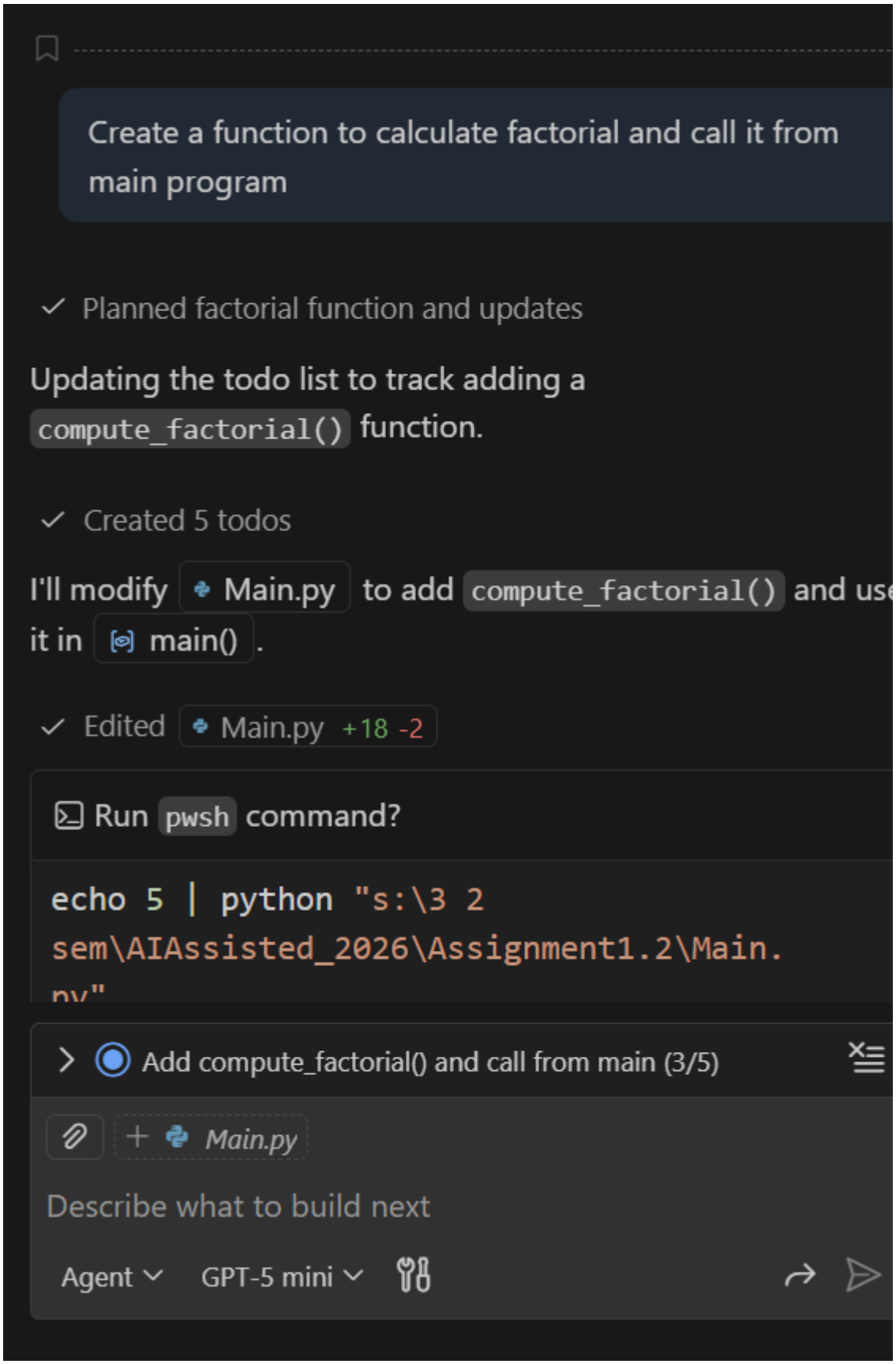
Main.py

S:\3 2 sem\AIAssisted_2026\Assignment1.2\Main.py

```python
def read_non_negative_int(prompt="Enter a non-negative integer: "):
    if n < 0:
        print("Factorial not defined for negative numbers.")
        sys.exit(1)
    return n


def main():
    n = read_non_negative_int()
    result = math.factorial(n)
    print(f"Factorial of {n} is {result}")


if __name__ == "__main__":
    main()

```

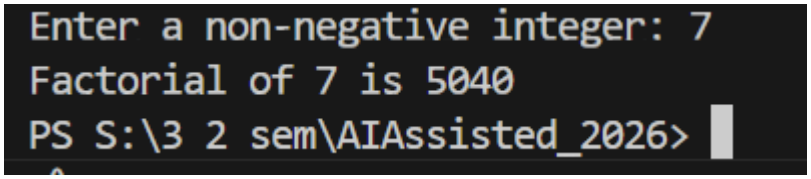
```
PS S:\3 2 sem\AIAssisted_2026> & C:/Users/sowmy/AppData/Local/Microsoft/WindowsApps/python3.11.exe "s:/3 2 sem
/AIAssisted_2026/Assignment1.2/Main.py"
Enter a non-negative integer: 7
Factorial of 7 is 5040
PS S:\3 2 sem\AIAssisted_2026>
```

TASK-3


Create a function to calculate factorial and call it from main program
Planned factorial function and updates
Updating the todo list to track adding a compute_factorial() function.
Created 5 todos
I'll modify Main.py to add compute_factorial() and use it in main() .
Edited Main.py +18 -2
Run pwsh command?
echo 5 | python "s:\3 2 sem\AIAssisted_2026\Assignment1.2\Main.py"
Add compute_factorial() and call from main (3/5)
Main.py
Describe what to build next
Agent
GPT-5 mini

```
  Improved for readability, correctness and performance by using
  the standard library `math.factorial` and a small input helper.
  """
  import sys


  def read_non_negative_int(prompt="Enter a non-negative integer: "):
      try:
          s = input(prompt)
      except EOFError:
          print("No input provided.")
          sys.exit(1)
      try:
          n = int(s)
      except ValueError:
          print("Invalid input. Please enter an integer.")
          sys.exit(1)
      if n < 0:
          print("Factorial not defined for negative numbers.")
          sys.exit(1)
      return n


  def compute_factorial(n: int) -> int:
      """Compute factorial iteratively to avoid recursion limits.

      Args:
          n: non-negative integer

```

```
  def compute_factorial(n: int) -> int:
      """Compute factorial iteratively to avoid recursion limits.

      Args:
          n: non-negative integer

      Returns:
          n! as int
      """
      if n < 2:
          return 1
      result = 1
      for i in range(2, n + 1):
          result *= i
      return result


  def main():
      n = read_non_negative_int()
      result = compute_factorial(n)
      print(f"Factorial of {n} is {result}")


  if __name__ == "__main__":
      main()
```



Modularity improves reusability because the same function can be used in multiple programs.
It also makes debugging easier and code more organized.

TASK-4

Comparative Analysis: Procedural vs Modular AI-Generated Code

AI-assisted coding has become an important tool in modern software development. In this experiment, two versions of the same logic were generated using GitHub Copilot: one written without functions (procedural) and the other using a modular function-based approach.

In the procedural version, all logic is written inside the main program block. While this approach is simple and easy to implement for small programs, the clarity of logic decreases as the program size increases. The absence of functions makes the code lengthy and difficult to manage. Reusability is very limited because the same code must be rewritten if required elsewhere. Debugging is also more difficult since all operations are mixed inside a single flow, making it harder to isolate errors.

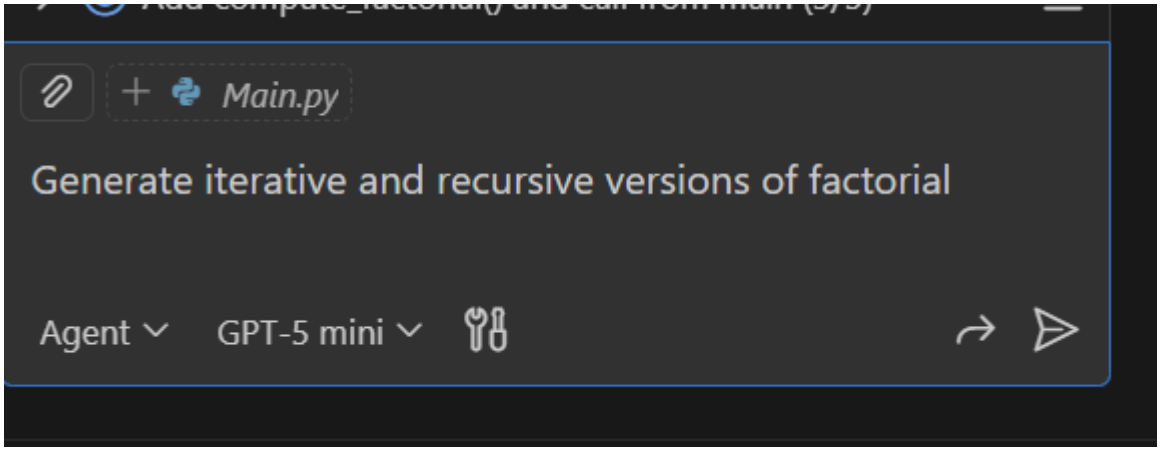
In contrast, the modular version organizes the logic inside a well-defined function. This significantly improves code clarity because each part of the program has a clear purpose. Reusability becomes a major advantage, as the same function can be reused in multiple programs without rewriting the code. Debugging becomes easier because errors can be traced within specific functions. This design is highly suitable for large projects where multiple developers work on the same codebase.

For large-scale applications, modular design is essential. It supports teamwork, version control, testing, and long-term maintenance. The procedural approach is only suitable for very small scripts or quick experimental programs.

AI dependency risk is higher in procedural code because errors in long logic blocks are harder to detect. Modular code reduces this risk by allowing developers to review and verify each function separately.

Therefore, modular AI-generated code is superior in terms of clarity, maintainability, reusability, debugging ease, and project scalability.

TASK-5




```python
import sys


def read_non_negative_int(prompt="Enter a non-negative integer: "):
    try:
        s = input(prompt)
    except EOFError:
        print("No input provided.")
        sys.exit(1)
    try:
        n = int(s)
    except ValueError:
        print("Invalid input. Please enter an integer.")
        sys.exit(1)
    if n < 0:
        print("Factorial not defined for negative numbers.")
        sys.exit(1)
    return n


def compute_factorial(n: int) -> int:
    """Compute factorial iteratively to avoid recursion limits.

    Args:
        n: non-negative integer

    Returns:
        n! as int
```

```python
    if n < 2:
        return 1
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result


def compute_factorial_recursive(n: int) -> int:
    """Compute factorial using recursion.

    Note: recursion depth may be reached for very large `n`.
    """
    if n < 2:
        return 1
    return n * compute_factorial_recursive(n - 1)


def main():
    n = read_non_negative_int()
    result = compute_factorial(n)
    print(f"Factorial of {n} is {result}")


if __name__ == "__main__":
    main()

```

```
Enter a non-negative integer: 7
Factorial of 7 is 5040
PS S:\3 2 sem\AIAssisted_2026>
```