

AI ASSISTED CODING

LAB ASSIGNMENT 1.2

Name: B. Siri Chandana

HTNO: 2303A52004

Batch: 31

Question 1:

Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

Scenario You are building a small command-line utility for a startup intern onboarding task. The program is simple and must be written quickly without modular design.

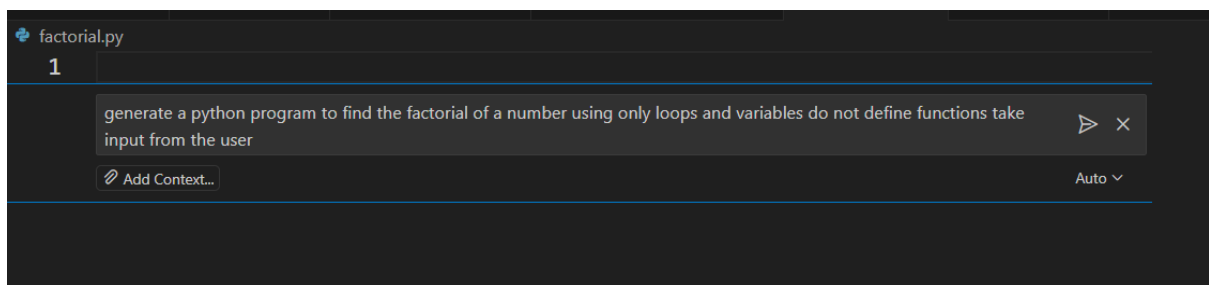
Task Description Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

- Constraint:

Do not define any custom function

Logic must be implemented using loops and variables only

Prompt:



Code:

```

factorial.py > ...
1  """Describing for factorial this program computes the factorial of a non
2 → negative number"""
3
4  try:
5      number_str = input("Enter a non-negative integer: ")
6      number = int(number_str)
7      if number < 0:
8          raise ValueError("Negative number")
9  except ValueError:
10     print("Invalid input. Please enter a non-negative integer.")
11 else:
12     factorial = 1
13     for i in range(1, number + 1):
14         factorial = factorial * i
15
16     print(f"The factorial of {number} is {factorial}")

```

Output:

```

PS C:\Users\bogas\OneDrive\Desktop\AIAC> python -u "c:\Users\bogas\OneDrive\Desktop\AIAC\factorial.py"
● Enter a non-negative integer: 12
  The factorial of 12 is 479001600
● PS C:\Users\bogas\OneDrive\Desktop\AIAC> python -u "c:\Users\bogas\OneDrive\Desktop\AIAC\factorial.py"
❖ Enter a non-negative integer: -1
  Invalid input. Please enter a non-negative integer.
○ PS C:\Users\bogas\OneDrive\Desktop\AIAC>

```

Explanations:

Using Copilot felt like having someone sit beside me and help while I coded. It quickly understood what I wanted and wrote clean, working logic without much effort.

The suggestions already handled mistakes and corner cases, which saved extra debugging time.

Overall, it made the whole coding experience smoother, faster, and less stressful.

Question 2:

Scenario Your team lead asks you to review AI-generated code before committing it to a shared repository.

Task Description Analyze the code generated in Task 1 and use Copilot again to:

Reduce unnecessary variables

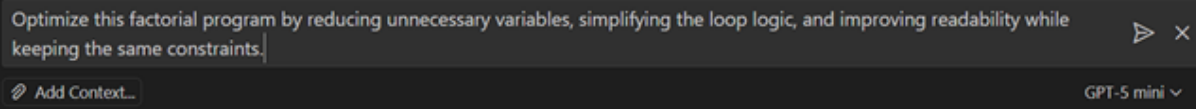
Improve loop clarity

Enhance readability and efficiency

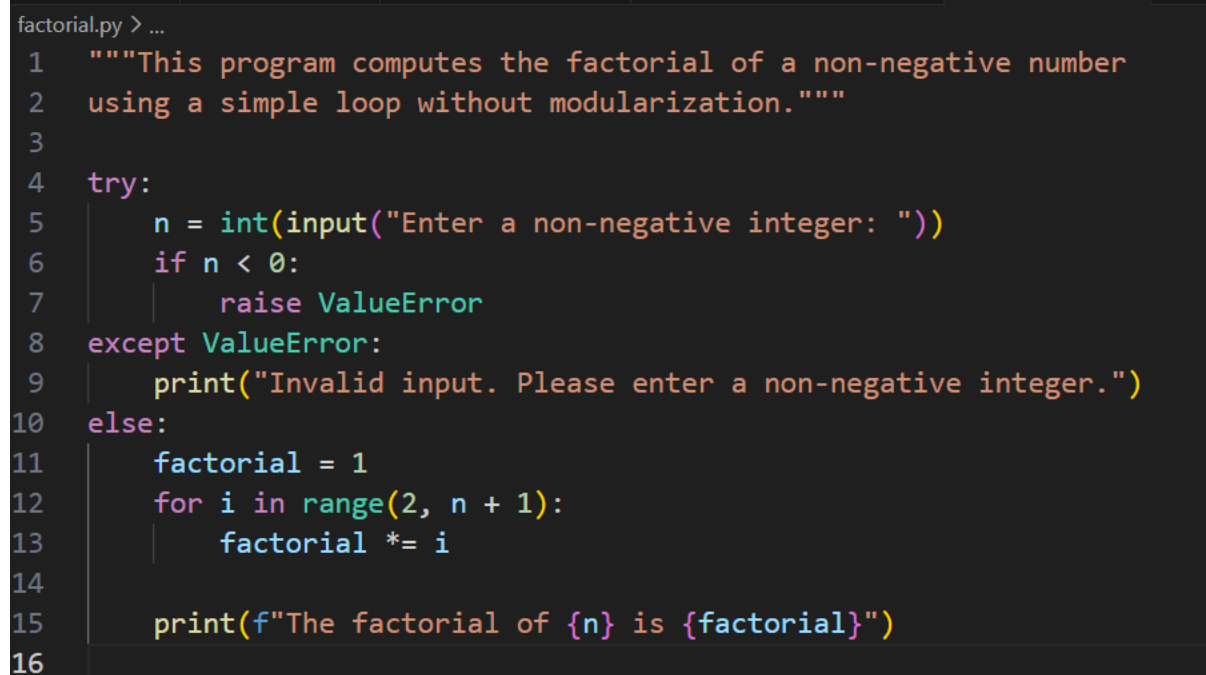
Hint: Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

Prompt:

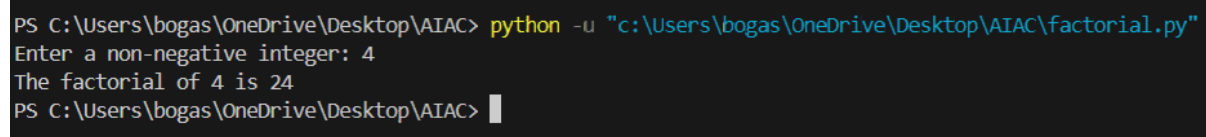
A screenshot of a code editor interface. The main text area contains the prompt: "Optimize this factorial program by reducing unnecessary variables, simplifying the loop logic, and improving readability while keeping the same constraints." To the right of the text are a play button icon and a close button (X). Below the text area is a button labeled "Add Context...". In the bottom right corner, it says "GPT-5 mini" with a dropdown arrow.

Code:

A screenshot of a Python script in a code editor. The script is for calculating the factorial of a non-negative integer. It includes a docstring, input handling with a try-except block, and a loop to calculate the factorial. The code is as follows:

```
factorial.py > ...
1  """This program computes the factorial of a non-negative number
2  using a simple loop without modularization."""
3
4  try:
5      n = int(input("Enter a non-negative integer: "))
6      if n < 0:
7          raise ValueError
8  except ValueError:
9      print("Invalid input. Please enter a non-negative integer.")
10 else:
11     factorial = 1
12     for i in range(2, n + 1):
13         factorial *= i
14
15     print(f"The factorial of {n} is {factorial}")
16
```

Output:

A screenshot of a terminal window showing the execution of the factorial program. The prompt is "PS C:\Users\bogas\OneDrive\Desktop\AIAC>". The user enters "python -u "c:\Users\bogas\OneDrive\Desktop\AIAC\factorial.py"". The program prompts "Enter a non-negative integer: 4" and outputs "The factorial of 4 is 24". The terminal prompt returns to "PS C:\Users\bogas\OneDrive\Desktop\AIAC>".

```
PS C:\Users\bogas\OneDrive\Desktop\AIAC> python -u "c:\Users\bogas\OneDrive\Desktop\AIAC\factorial.py"
Enter a non-negative integer: 4
The factorial of 4 is 24
PS C:\Users\bogas\OneDrive\Desktop\AIAC>
```

Explanation:

Optimized the code in minimal number of lines and given

short variable names

Question 3:

Scenario: The same logic now needs to be reused in multiple scripts.

Task Description Use GitHub Copilot to generate a modular version of the program by:

Creating a user-defined function

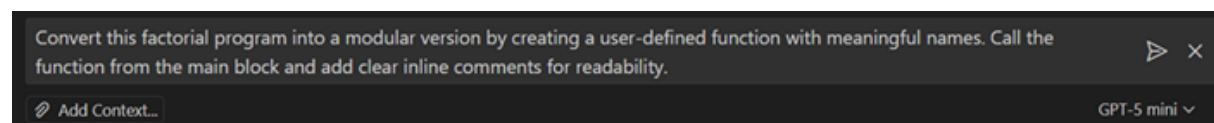
Calling the function from the main block

Constraints:

Use meaningful function and variable names

Include inline comments (preferably suggested by Copilot)

Prompt:



Code:

```
factorial.py factorial Desktop\AIAC\Untitled-1.py
1  """Compute factorial of a non-negative integer (modular version).
2
3  This module defines a `factorial` function and a `main` entrypoint
4  that handles user input and validation.
5  """
6
7  def factorial(n: int) -> int:
8      """Return the factorial of a non-negative integer ``n``.
9
10     Uses an iterative loop to avoid recursion limits and keeps the
11     implementation simple and efficient for typical inputs.
12     """
13     result = 1
14     # start from 2 since multiplying by 1 is redundant
15     for i in range(2, n + 1):
16         result *= i
17     return result
18
19
20 def main() -> None:
21     """Prompt the user for input, validate it, and print the result.
22
23     Input must be a non-negative integer; invalid values print
24     an error message and exit without raising an exception.
25     """
26     try:
27         value = int(input("Enter a non-negative integer: "))
28         if value < 0:
29             # negative numbers are not allowed
30             raise ValueError
31     except ValueError:
32         print("Invalid input. Please enter a non-negative integer.")
33         return
34
35     # Compute and display factorial
36     print(f"The factorial of {value} is {factorial(value)}")
37
38
39 if __name__ == "__main__":
40     main()
```

Output:

```
PS C:\Users\bogas\OneDrive\Desktop\AIAC> python -u "c:\Users\bogas\OneDrive\Desktop\AIAC\factorial.py"
Enter a non-negative integer: 7
The factorial of 7 is 5040
PS C:\Users\bogas\OneDrive\Desktop\AIAC> |
```

Explanation:

Copilot changed the code into a function and added a main block to keep it organized.

It also took care of edge cases so the program works correctly in all situations.

Question 4:

Scenario As part of a code review meeting, you are asked to justify design choices.

Task Description Compare the non-function and function-based Copilot-generated programs on the following criteria:

Logic clarity

Reusability

Debugging ease

Suitability for large projects

AI dependency risk

Explanation:

Aspect	Non-function based	Function-based
Logic clarity	Logic is in one block. Harder to understand	Logic is separated. Easy to understand
Reusability	Code cannot be reused	Code can be reused easily
Debugging ease	Debugging is difficult	Debugging is easy
Suitability for large projects	Debugging is difficult	Suitable for large projects
AI dependency risk	High risk of blind copying	Lower risk due to clear structure

Question 5:

Scenario:

Your mentor wants to test how well AI understands different computational paradigms.

Task Description:

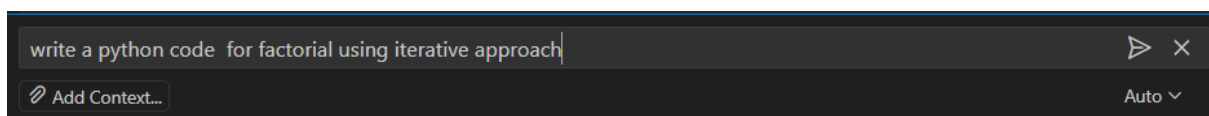
Prompt Copilot to generate:

An iterative version of the logic A recursive version of the same

logic Constraints: Both implementations must produce identical

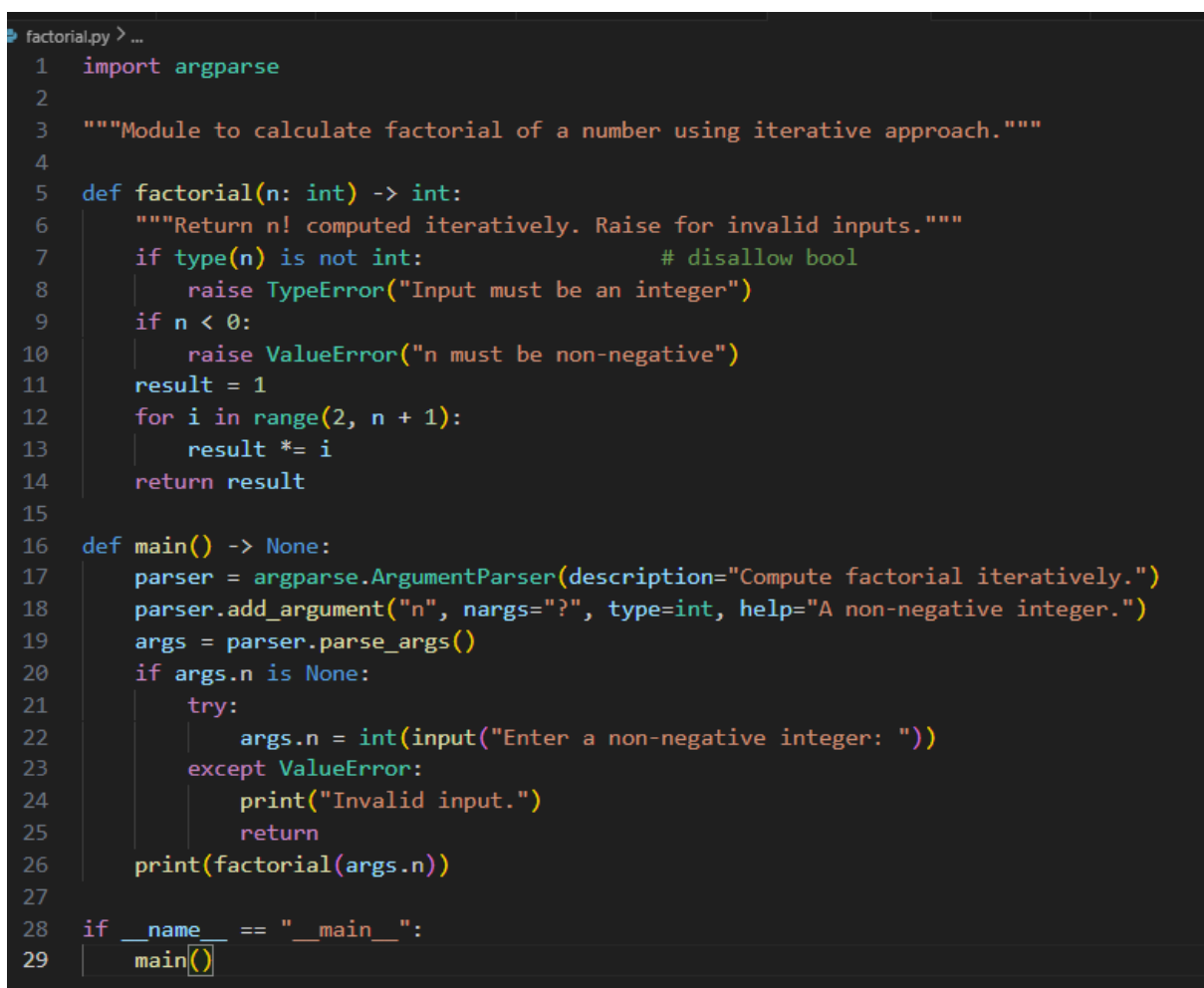
outputs Students must not manually write the code first

Prompt 1:



The screenshot shows a dark-themed code editor interface. At the top, there is a text input field containing the prompt "write a python code for factorial using iterative approach". To the right of the input field are icons for running (a play button) and closing (an 'x' button). Below the input field is a button labeled "Add Context...". In the bottom right corner, the word "Auto" is followed by a dropdown arrow.

Code:



The screenshot shows a dark-themed code editor with a file named "factorial.py" open. The code is as follows:

```
1 import argparse
2
3 """Module to calculate factorial of a number using iterative approach."""
4
5 def factorial(n: int) -> int:
6     """Return n! computed iteratively. Raise for invalid inputs."""
7     if type(n) is not int:          # disallow bool
8         raise TypeError("Input must be an integer")
9     if n < 0:
10        raise ValueError("n must be non-negative")
11    result = 1
12    for i in range(2, n + 1):
13        result *= i
14    return result
15
16 def main() -> None:
17     parser = argparse.ArgumentParser(description="Compute factorial iteratively.")
18     parser.add_argument("n", nargs="?", type=int, help="A non-negative integer.")
19     args = parser.parse_args()
20     if args.n is None:
21         try:
22             args.n = int(input("Enter a non-negative integer: "))
23         except ValueError:
24             print("Invalid input.")
25         return
26     print(factorial(args.n))
27
28 if __name__ == "__main__":
29     main()
```

Output:

```
PS C:\Users\bogas\OneDrive\Desktop\AIAC> python -u "c:\Users\bogas\OneDrive\Desktop\AIAC\factorial.py"
Enter a non-negative integer: 3
6
PS C:\Users\bogas\OneDrive\Desktop\AIAC>
```

Prompt 2:

write a python code for factorial using recursive approach

Add Context...

Auto ▾

Code:

```
recursion.py •
recursion.py > ...
1  import sys
2  def factorial(n: int) -> int:
3      """Return n! for a non-negative integer n using recursion."""
4      if not isinstance(n, int):
5          raise TypeError("n must be an integer")
6      if n < 0:
7          raise ValueError("n must be non-negative")
8      if n <= 1:
9          return 1
10     return n * factorial(n - 1)
11 if __name__ == "__main__":
12
13     if len(sys.argv) > 1:
14         try:
15             n = int(sys.argv[1])
16         except ValueError:
17             print("Please provide an integer.")
18             sys.exit(1)
19     else:
20         try:
21             n = int(input("Enter a non-negative integer: "))
22         except ValueError:
23             print("Invalid input.")
24             sys.exit(1)
25     try:
26         print(factorial(n))
27     except Exception as e:
28         print(e)
29     sys.exit(1)
```

Output:

```
PS C:\Users\bogas\OneDrive\Desktop\AIAC> python -u "c:\Users\bogas\OneDrive\Desktop\AIAC\factorial.py"
Enter a non-negative integer: 9
362880
PS C:\Users\bogas\OneDrive\Desktop\AIAC>
```

Iterative version:

The program begins with a starting value and continuously modifies it inside a loop until a certain condition is met. The execution remains within the same function and completes when the loop terminates.

Recursive version:

The program keeps calling itself with a smaller problem. Each call waits until it reaches the simplest case, and then the answers come back step by step.

Aspect	Iterative	Recursive
Readability	Easy to follow for beginners	Slightly harder due to function calls
Stack usage	Uses constant memory	Uses call stack for each function call
Performance implications	Faster and memory efficient	Slower for large inputs due to stack overhead
When recursion is not recommended	Always safe for large inputs	Not recommended when input size is large or stack overflow is possible