ASSIGNMENT-1.2

2303A51803

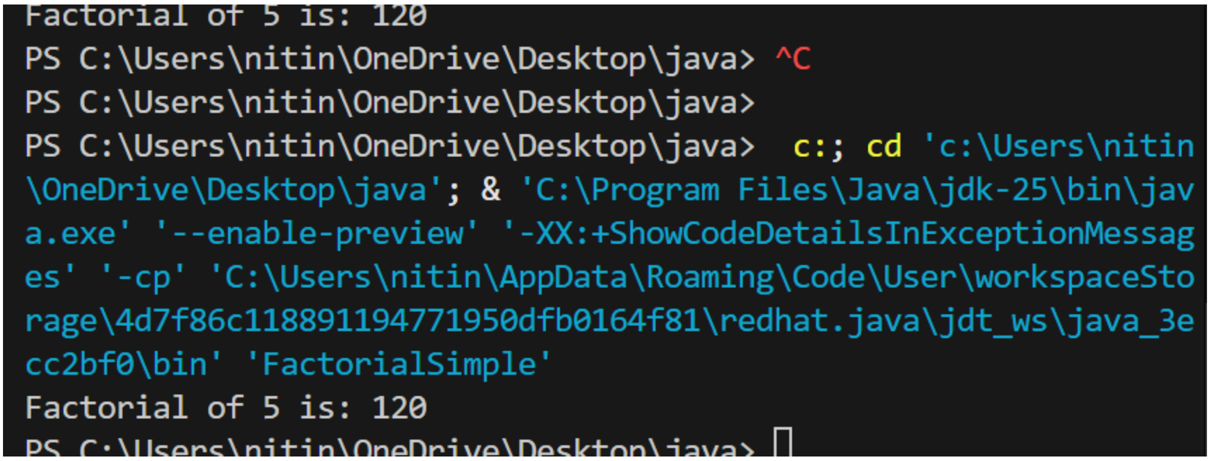
Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

Prompt:

create a java file to print factorial number using for loop and variables

code:

```
public class FactorialSimple {
   public static void main(String[] args) {
      int number = 5; // The number to find factorial for
      long factorial = 1; // Variable to store the result

      // Using for loop to calculate factorial
      for (int i = 1; i <= number; i++) {
         factorial = factorial * i;
      }

      System.out.println("Factorial of " + number + " is: " + factorial);
   }
}
```



```
Factorial of 5 is: 120
PS C:\Users\nitin\OneDrive\Desktop\java> ^C
PS C:\Users\nitin\OneDrive\Desktop\java>
PS C:\Users\nitin\OneDrive\Desktop\java>  c:; cd 'c:\Users\nitin\OneDrive\Desktop\java'; & 'C:\Program Files\Java\jdk-25\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\nitin\AppData\Roaming\Code\User\workspaceStorage\4d7f86c118891194771950dfb0164f81\redhat.java\jdt_ws\java_3ecc2bf0\bin' 'FactorialSimple'
Factorial of 5 is: 120
PS C:\Users\nitin\OneDrive\Desktop\java>
```

Undestanding:

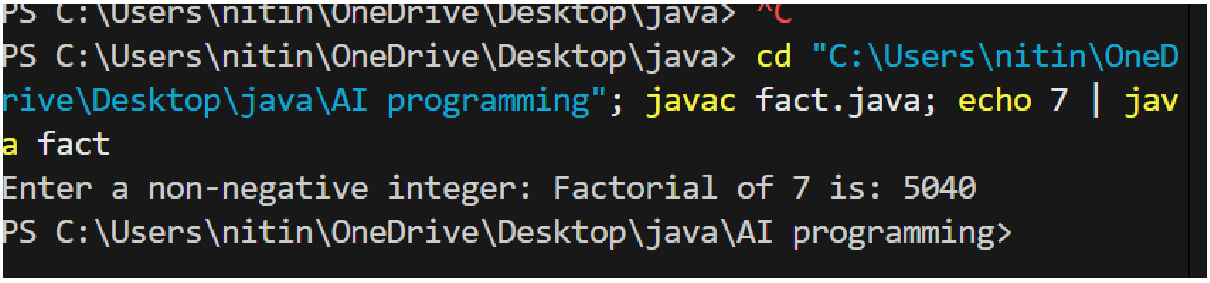
From this task, I understood how GitHub Copilot helps in quickly writing code without much effort.
It generated a correct factorial program using loops and variables without any functions.
This made it easier for me to learn how AI can support beginners in coding.
I also understood how useful Copilot is when we need fast and working solutions.

Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

Prompt: // Optimized version of FactorialOfANumber.java remove unnecessary variable and loops

Code:

```
import java.math.BigInteger;
import java.util.Scanner;

public class fact {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a non-negative integer: ");
        if (!sc.hasNextInt()) {
            System.out.println("Invalid input");
            sc.close();
            return;
        }
        int n = sc.nextInt();
    sc.close();

    if (n < 0) {

      System.out.println("Factorial not defined for negative numbers");

      return;

    }

---

    BigInteger result = BigInteger.ONE;

    for (int i = 2; i <= n; i++) {

      result = result.multiply(BigInteger.valueOf(i));

    }


    System.out.println("Factorial of " + n + " is: " + result);

  }

}
```



```
PS C:\Users\nitin\OneDrive\Desktop\java> ^C
PS C:\Users\nitin\OneDrive\Desktop\java> cd "C:\Users\nitin\OneD
rive\Desktop\java\AI programming"; javac fact.java; echo 7 | jav
a fact
Enter a non-negative integer: Factorial of 7 is: 5040
PS C:\Users\nitin\OneDrive\Desktop\java\AI programming>
```

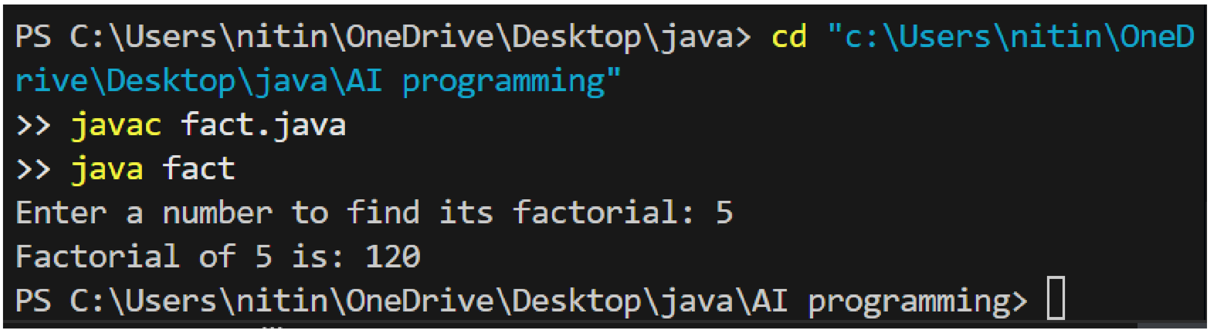
Here I learned how Copilot can also improve and clean the code, not just generate it.
It helped in removing unnecessary parts and made the program more efficient and readable.
I understood why optimized code is important in real projects.
This task taught me how AI can help in writing professional and cleaner code.

Task 3: Modular Design Using AI Assistance (Factorial with Functions)

Prompt: //create a function to calculate factorial of a number and call it in main method

Code:

```
//create a Java program to calculate the factorial of a number using recursion


import java.util.Scanner;   public class FactorialRecursive {


  // Method to calculate factorial using recursion
  public static long factorial(int n) {
    if (n == 0 || n == 1) {
      return 1; // Base case: factorial of 0 or 1 is 1
    } else {
      return n * factorial(n - 1); // Recursive case
    }
  }


  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);


    System.out.print("Enter a number to find its factorial: ");
    int number = scanner.nextInt();


    long result = factorial(number);
    System.out.println("Factorial of " + number + " is: " + result);


    scanner.close();
  }
}
```



```
PS C:\Users\nitin\OneDrive\Desktop\java> cd "c:\Users\nitin\OneDrive\Desktop\java\AI programming"
>> javac fact.java
>> java fact
Enter a number to find its factorial: 5
Factorial of 5 is: 120
PS C:\Users\nitin\OneDrive\Desktop\java\AI programming> 
```

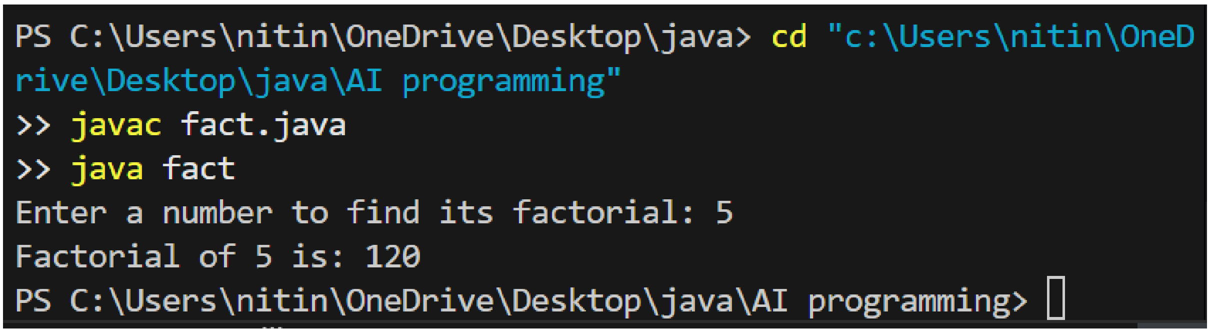
In this task, I understood what modular programming means and why functions are useful.
Using a function makes the program more organized and easier to understand.
It also helps in reusing the same logic in other programs without rewriting it.
Copilot helped in creating a neat and structured program with proper logic.

Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs

Without Functions)

Code: code with using function

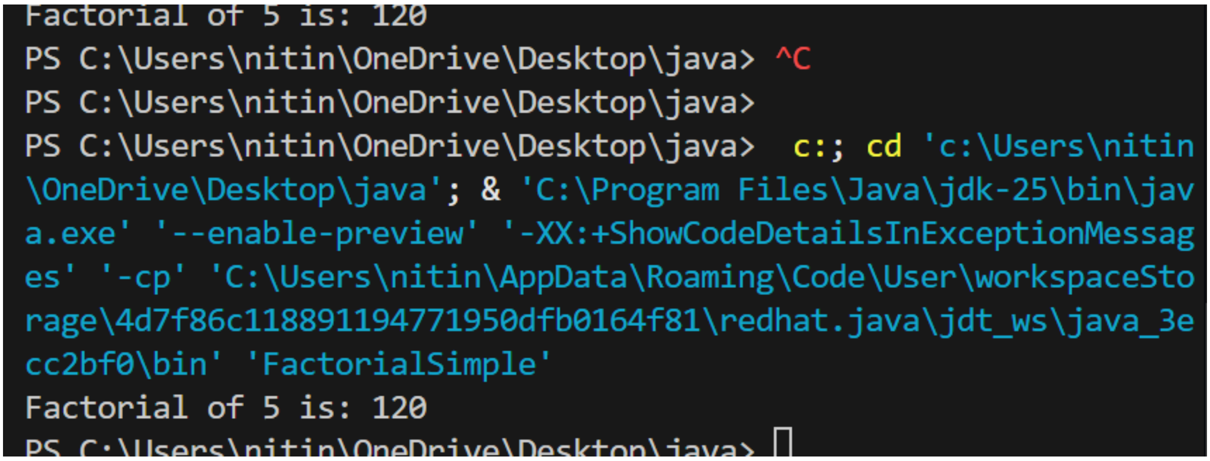
```
import java.util.Scanner;   public class FactorialRecursive {


  // Method to calculate factorial using recursion

  public static long factorial(int n) {

    if (n == 0 || n == 1) {

      return 1; // Base case: factorial of 0 or 1 is 1
    } else {
      return n * factorial(n - 1); // Recursive case
    }
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter a number to find its factorial: ");
    int number = scanner.nextInt();

    long result = factorial(number);
    System.out.println("Factorial of " + number + " is: " + result);

    scanner.close();
  }
}
```



```
PS C:\Users\nitin\OneDrive\Desktop\java> cd "c:\Users\nitin\OneD
rive\Desktop\java\AI programming"
>> javac fact.java
>> java fact
Enter a number to find its factorial: 5
Factorial of 5 is: 120
PS C:\Users\nitin\OneDrive\Desktop\java\AI programming>
```

Code : code without using function

```
public class FactorialSimple {
  public static void main(String[] args) {
    int number = 5; // The number to find factorial for
    long factorial = 1; // Variable to store the result
    // Using for loop to calculate factorial
    for (int i = 1; i <= number; i++) {
      factorial = factorial * i;
    }

    System.out.println("Factorial of " + number + " is: " + factorial);
  }
}
```



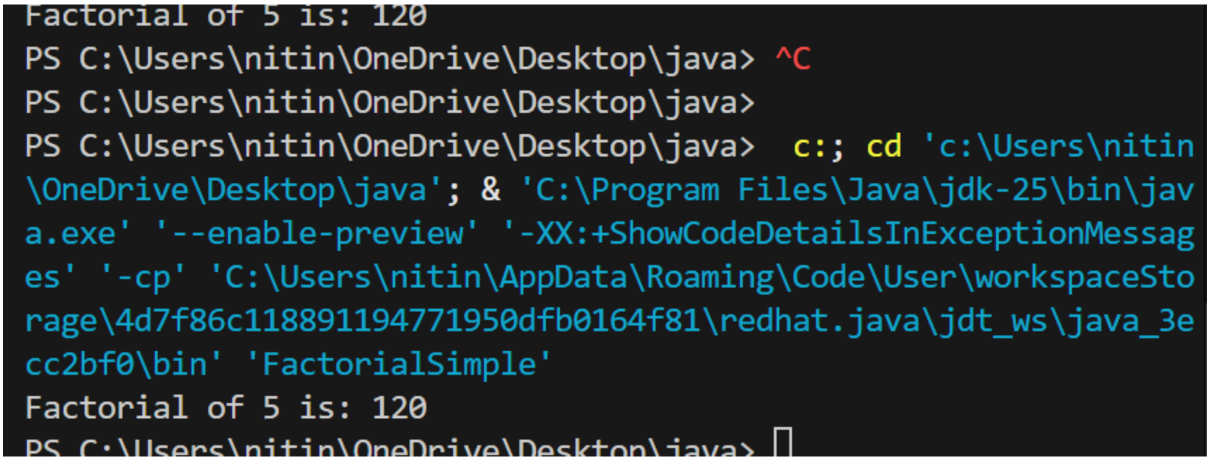
```
Factorial of 5 is: 120
PS C:\Users\nitin\OneDrive\Desktop\java> ^C
PS C:\Users\nitin\OneDrive\Desktop\java>
PS C:\Users\nitin\OneDrive\Desktop\java>  c:; cd 'c:\Users\nitin
\OneDrive\Desktop\java'; & 'C:\Program Files\Java\jdk-25\bin\jav
a.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessag
es' '-cp' 'C:\Users\nitin\AppData\Roaming\Code\User\workspaceSto
rage\4d7f86c118891194771950dfb0164f81\redhat.java\jdt_ws\java_3e
cc2bf0\bin' 'FactorialSimple'
Factorial of 5 is: 120
PS C:\Users\nitin\OneDrive\Desktop\java> 
```

From this task, I learned the difference between writing code with and without functions.
Code without functions is simple but not suitable for bigger projects.
Code with functions is clearer, reusable, and easier to debug and manage.
This helped me understand why modular programming is preferred in real-world coding.

Task 5: AI-Generated Iterative vs Recursive Thinking

Code: code AI-generative Interative

```
public class FactorialSimple {
  public static void main(String[] args) {
    int number = 5; // The number to find factorial for
    long factorial = 1; // Variable to store the result

    // Using for loop to calculate factorial
    for (int i = 1; i <= number; i++) {
      factorial = factorial * i;
    }

    System.out.println("Factorial of " + number + " is: " + factorial);
  }
}
```



```
Factorial of 5 is: 120
PS C:\Users\nitin\OneDrive\Desktop\java> ^C
PS C:\Users\nitin\OneDrive\Desktop\java>
PS C:\Users\nitin\OneDrive\Desktop\java>  c:; cd 'c:\Users\nitin
\OneDrive\Desktop\java'; & 'C:\Program Files\Java\jdk-25\bin\jav
a.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessag
es' '-cp' 'C:\Users\nitin\AppData\Roaming\Code\User\workspaceSto
rage\4d7f86c118891194771950dfb0164f81\redhat.java\jdt_ws\java_3e
cc2bf0\bin' 'FactorialSimple'
Factorial of 5 is: 120
PS C:\Users\nitin\OneDrive\Desktop\java> 
```

Code: with recursion

```
import java.util.Scanner;   public class FactorialRecursive {

  // Method to calculate factorial using recursion
  public static long factorial(int n) {
    if (n == 0 || n == 1) {
      return 1; // Base case: factorial of 0 or 1 is 1
    } else {
      return n * factorial(n - 1); // Recursive case
    }
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter a number to find its factorial: ");
    int number = scanner.nextInt();

    long result = factorial(number);
    System.out.println("Factorial of " + number + " is: " + result);

    scanner.close();
```

In this task, I understood the difference between iterative and recursive methods.
Iteration uses loops and is generally faster and memory-friendly.
Recursion looks simple but uses more stack memory and may be risky for large inputs.
This helped me understand when we should and should not use recursion.