

AI ASSISTED CODING

ASSIGNMENT

Name: G Hasini

Roll no: 2503A51L41

Batch: 24BTCAICSB20

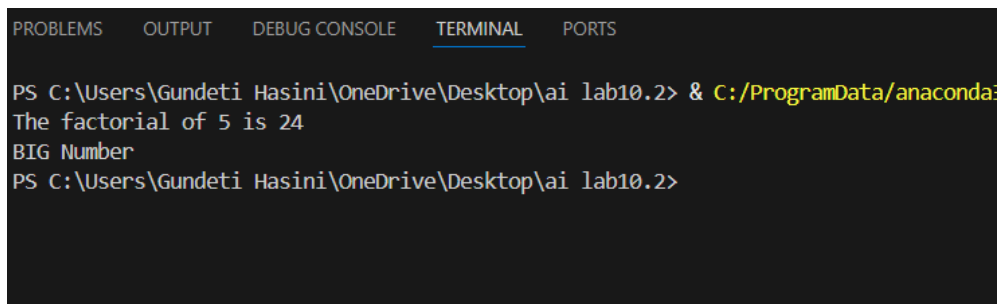
Lab 10: Code Review and Quality: Using AI to improve code quality and readability

Task#1: AI-Assisted Code Review (Basic Errors)

- Write python program as shown below.

```
def calcFact(n):  
    result=1  
    x=0  
    for i in range(1,n):  
        result=result*i  
    return result  
  
def main():  
    num = 5  
    FACT = calcFact(num)  
    print("the factorial of",num,"is",FACT)  
    t=10  
    if FACT>10:  
        print("BIG Number")  
    else:  
        print("small number")  
  
main()
```

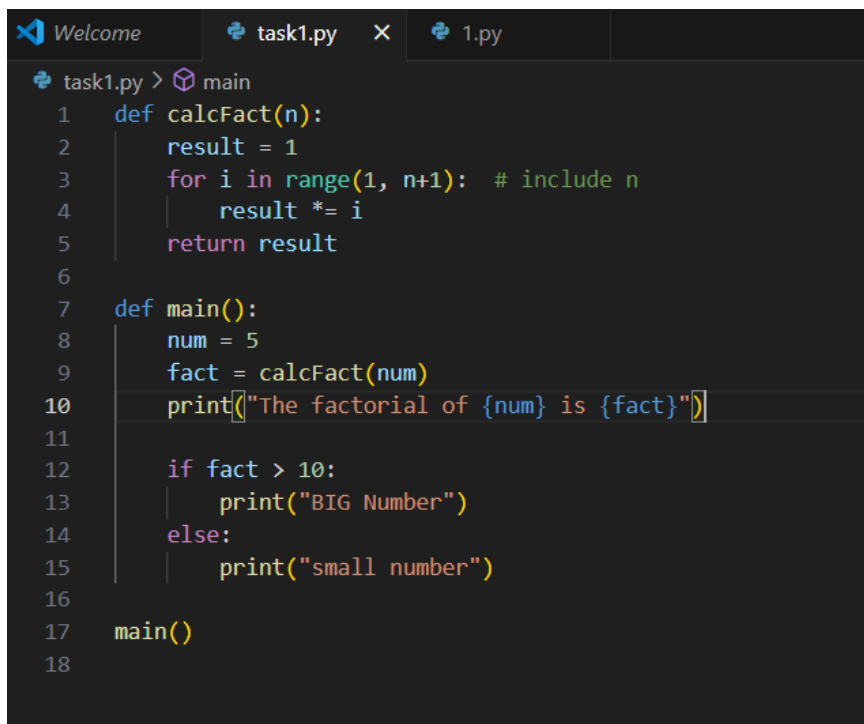
Output:



The screenshot shows a code editor with a dark theme. At the top, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active. The terminal output shows the command 'PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2> & C:/ProgramData/anaconda...' followed by the program's output: 'The factorial of 5 is 24' and 'BIG Number'. The prompt 'PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2>' is shown again at the bottom.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2> & C:/ProgramData/anaconda  
The factorial of 5 is 24  
BIG Number  
PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2>
```

Prompt: Use an AI assistant to review and suggest corrections

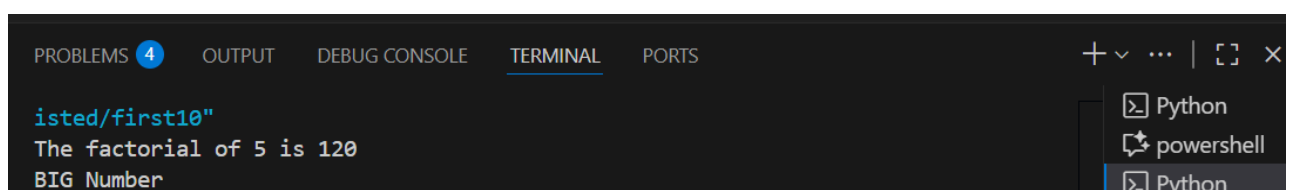


```
Welcome task1.py x 1.py
task1.py > main
1 def calcFact(n):
2     result = 1
3     for i in range(1, n+1): # include n
4         result *= i
5     return result
6
7 def main():
8     num = 5
9     fact = calcFact(num)
10    print("The factorial of {num} is {fact}")
11
12    if fact > 10:
13        print("BIG Number")
14    else:
15        print("small number")
16
17    main()
18
```

Issues:

1. Factorial calculation range
 - Currently: for i in range(1,n):
 - This loop runs from 1 to n-1, so it misses multiplying by n.
2. Unused variable
 - x=0 inside calcFact is not used at all. It can be removed.
3. Unnecessary variable t=10
4. You define t=10 but never use it. Code readability
 - Variable FACT should ideally be lowercase (fact) to follow Python naming conventions.

Output:



```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS
isted/first10"
The factorial of 5 is 120
BIG Number
```

Task#2: Automatic Inline Comments

- Write the Python code for Fibonacci as shown below.
- Ask AI to improve variable names, add comments, and apply PEP8 formatting (cleaned up).

```
def f1(xX):
    a=0
    b=1
    c=2
    Zz=[a,b]
    while c<=xX:
        d=a+b
        Zz.append(d)
        a=b
        b=d
        c=c+1
    return Zz

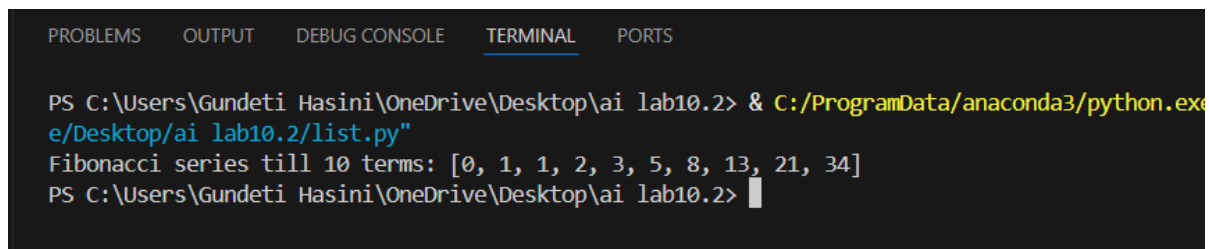
def m():
    NN=10
    ans=f1(NN)
    print("fib series till",NN,":",ans)

m()
```

Prompt: Ask AI to improve variable names, add comments, and apply PEP8 formatting (cleaned up).

```
second10.py > ...
1  # Function to generate Fibonacci series up to 'n' terms
2  def generate_fibonacci_series(n_terms):
3      first = 0
4      second = 1
5      count = 2 # Starts from 2 since 0 and 1 are already added
6      series = [first, second]
7      # Generate the rest of the series
8      while count < n_terms:
9          next_number = first + second
10         series.append(next_number)
11         first = second
12         second = next_number
13         count += 1
14     return series
15 # Main function
16 def main():
17     number_of_terms = 10
18     fibonacci_series = generate_fibonacci_series(number_of_terms)
19     print("Fibonacci series till", number_of_terms, "terms:", fibonacci_series)
20 # Execute main
21 main()
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Gundet Hasini\OneDrive\Desktop\ai lab10.2> & C:/ProgramData/anaconda3/python.exe
e/Desktop/ai lab10.2/list.py"
Fibonacci series till 10 terms: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\Gundet Hasini\OneDrive\Desktop\ai lab10.2> |
```

Key Improvements:

- Replacing a, b, c, Zz with first, second, count, series makes the logic self-explanatory.
- Function name generate_fibonacci_series tells you exactly what the function does.
- Original used while c <= xX, which actually **generates xX + 1 terms**, but your corrected code uses while count < n_terms, which properly limits the list to **exactly n_terms values**.
- Each logical block now has a comment — helps learners or future readers quickly understand what the code is doing.
- PEP8 style: consistent spacing, indentation, and line length.
- Your code looks professional and beginner-friendly now.

Task#3

Prompt: Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).

- Incorporate manual **docstring** in code with NumPy Style
- Use AI assistance to generate a module-level docstring + individual function docstrings.
- Compare the AI-generated docstring with your manually written one.

Manual code:

```
third10man.py > ...
1  def add(a, b):
2      """
3      Add two numbers.
4      args
5      a : The first number.
6      b : The second number.
7      Returns The sum of a and b.
8      """
9      return a + b
10 def subtract(a, b):
11     """
12     Subtract one number from another.
13     args
14     a : The number to subtract from.
15     b : The number to subtract.
16     Returns The result of a - b.
17     """
18     return a - b
19 def multiply(a, b):
20     """
21     Multiply two numbers.
22     args
23     a : The first number.
24     b : The second number.
25     Returns The product of a and b.
26     """
27     return a * b
28 def divide(a, b):
29     """
30     Divide one number by another.
31     args
32     a : The numerator.
33     b : The denominator.
34     Returns The result of a / b.
35     Raises ZeroDivisionError
36     |   If b is zero.
37     """
38     if b == 0:
39         raise ZeroDivisionError("Cannot divide by zero.")
40     return a / b
41
```

AI generated code:

```
third10ai.py > ...
1  """
2  basic_calculator.py
3  A simple calculator module that provides basic arithmetic operations:
4  addition, subtraction, multiplication, and division.
5  This module is intended for demonstration or educational use,
6  showcasing the use of modular functions and NumPy-style docstrings.
7  Functions
8  """
9  - add(a, b)
10 - subtract(a, b)
11 - multiply(a, b)
12 - divide(a, b)
13 """
14 def add(a, b):
15     """
16     Add two numbers.
17     Parameters
18     """
19     a : int or float
20     | The first number.
21     b : int or float
22     | The second number.
23     Returns
24     """
25     int or float
26     | The sum of a and b.
27     """
28     return a + b
29 def subtract(a, b):
30     """
31     Subtract one number from another.
32     Parameters
33     """
34     a : int or float
35     | The number to subtract from.
36     b : int or float
37     | The number to subtract.
38     Returns
39     """
40     int or float
41     | The result of a - b.
42     """
43     return a - b
44 def multiply(a, b):
45     """
46     Multiply two numbers.
47     Parameters
48     """
49     a : int or float
50     | The first number.
51     b : int or float
52     | The second number.
53     Returns
54     """
55     int or float
56     | The product of a and b.
57     """
58     return a * b
59 def divide(a, b):
60     """
61     Divide one number by another.
62     Parameters
63     """
64     a : int or float
65     | The numerator.
66     b : int or float
67     | The denominator.
68     Returns
69     """
70     float
71     | The result of a / b.
72     Raises
73     """
74     ZeroDivisionError
75     | If the denominator is zero.
76     """
77     if b == 0:
78         raise ZeroDivisionError("Cannot divide by zero.")
79     return a / b
80
```

Observation:

AI-assisted docstrings offer:

- Professional structure
- Better readability
- Tool compatibility
- Reusability in teams, APIs, and documentation websites

Manual comments are a good starting point for beginners, but lack structure and formal style, which may be insufficient for large or shared codebases.

The **manual comments** provide basic understanding but lack structure, standard formatting, and detailed type information. In contrast, the **AI-assisted docstrings** follow the **NumPy documentation standard**, offering clearer structure, better readability, and compatibility with professional tools. This makes them more suitable for collaborative and scalable projects.

