SR UNIVERSITY

AI ASSIST CODING

Lab 7: Error Debugging with AI – Systematic Approaches to Finding and Fixing Bugs

NAME: SHAIK FAHEEM

HTNO :2503A51L39

Lab Objectives:

- To identify and correct syntax, logic, and runtime errors in Python programs using AI tools.
- To understand common programming bugs and AI-assisted debugging suggestions.
- To evaluate how AI explains, detects, and fixes different types of coding errors.
- To build confidence in using AI to perform structured debugging practices.

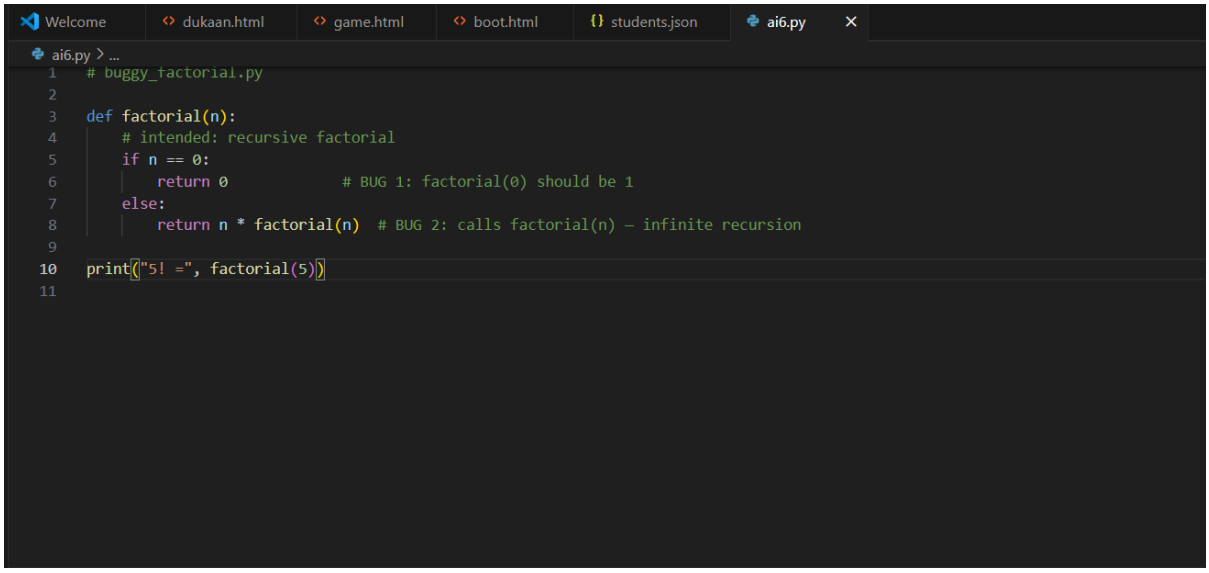
Lab Outcomes (LOs):

After completing this lab, students will be able to:

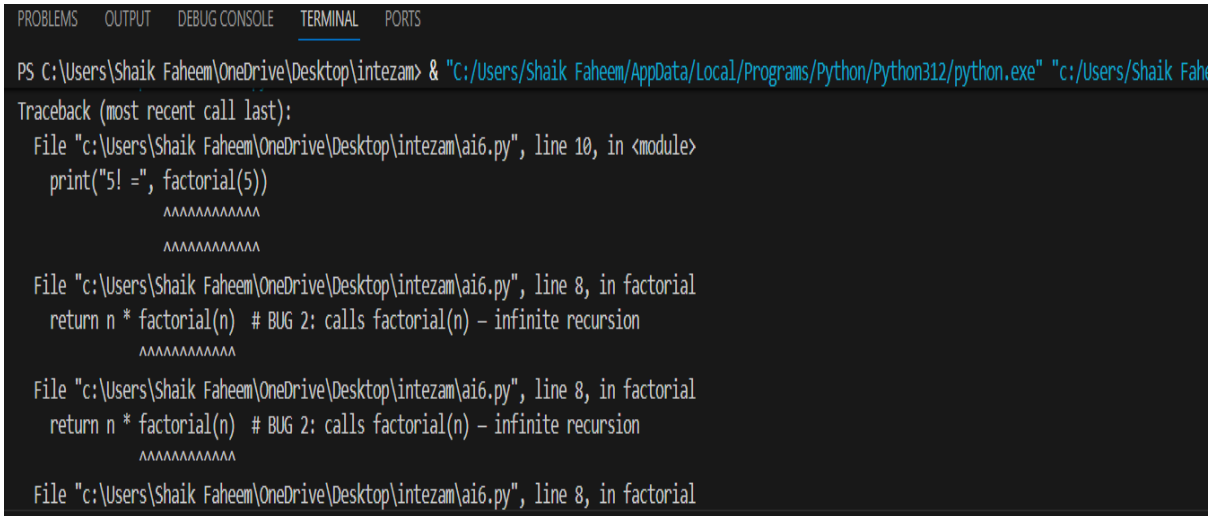
- Use AI tools to detect and correct syntax, logic, and runtime errors.
- Interpret AI-suggested bug fixes and explanations.
- Apply systematic debugging strategies supported by AI-generated insights.
- Refactor buggy code using responsible and reliable programming patterns.

TASK#1 : Introduce a buggy Python function that calculates the factorial of a number using recursion. Use Copilot or Cursor AI to detect and fix the logical or syntax errors

Bugged program given to the ai:

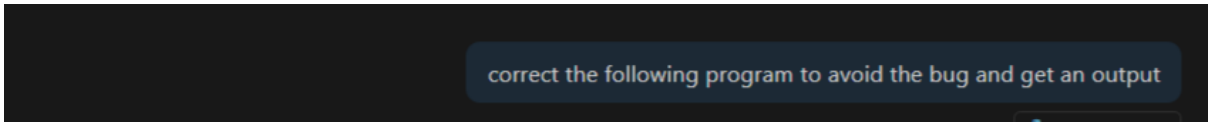


Output of bugged program:



given prompt to the AI:

Correct the following program to avoid the bug and give an output



given program by the AI:

```
def factorial(n):
        raise TypeError("factorial() only accepts integers")
    if n < 0:
        raise ValueError("factorial() not defined for negative numbers")

    # base cases
    if n == 0 or n == 1:
        return 1

    # recursive step (decrease n)
    return n * factorial(n - 1)

if __name__ == "__main__":
    print("0! =", factorial(0))   # 1
    print("1! =", factorial(1))   # 1
    print("5! =", factorial(5))   # 120
```

Final output after execution:

```
PS C:\Users\Shaik Faheem\OneDrive\Desktop\intezam> & "C:/Users/Shaik Faheem/AppData/Local/Programs/Python/Python312/python.exe" "c:/
OneDrive/Desktop/intezam/ai6.py"
0! = 1
1! = 1
5! = 120
PS C:\Users\Shaik Faheem\OneDrive\Desktop\intezam>
```

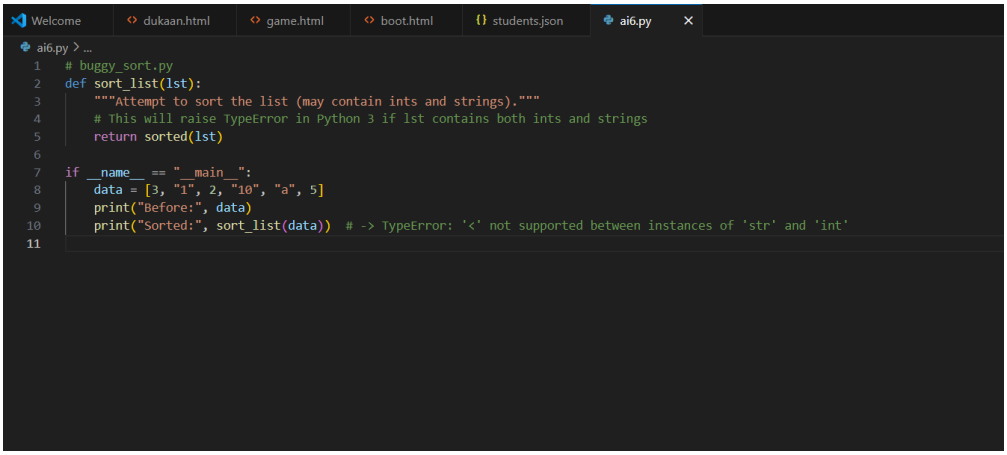
Observation:

The buggy factorial function causes infinite recursion due to an incorrect or missing base case, which Copilot/Cursor AI can detect and fix.

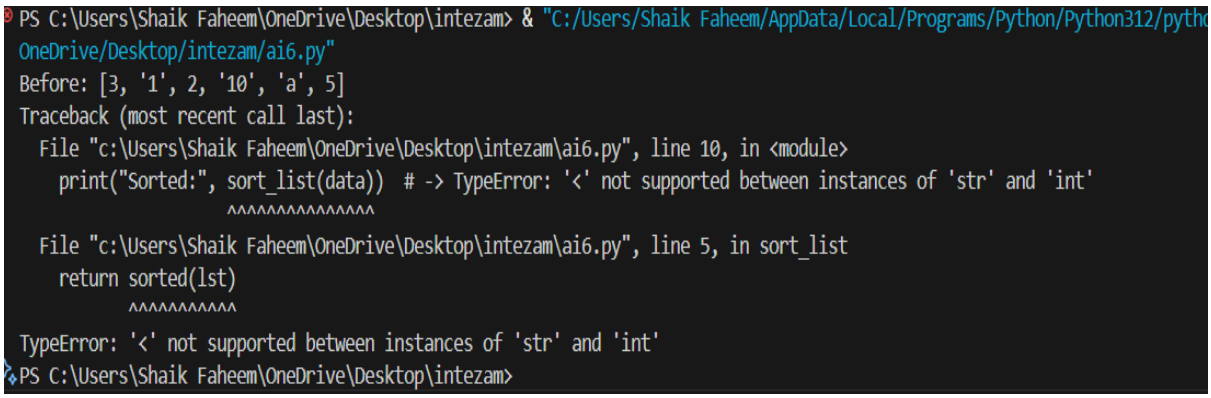
Task #2:

Provide a list sorting function that fails due to a type error (e.g., sorting list with mixed integers and strings). Prompt AI to detect the issue and fix the code for consistent sorting

Bugged program given to the ai:

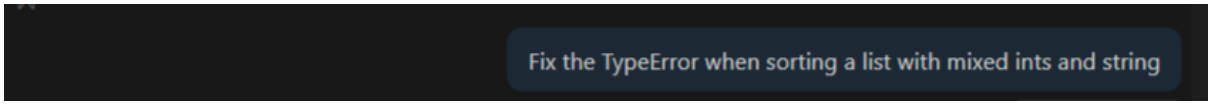


Output of the bugged program:

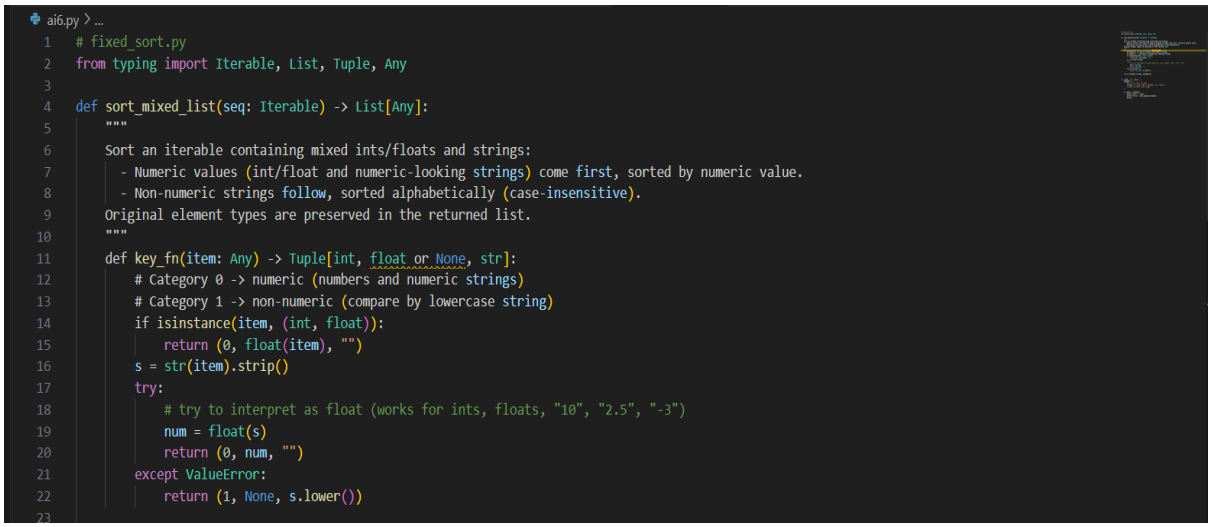


Given prompt to the ai:

fix the typeError when sorting a list with mixed ints and strings



Given program by the AI:



final output of the program:

```
OneDrive/Desktop/intezam/ai6.py
PS C:\Users\Shaik Faheem\OneDrive\Desktop\intezam> & "C:/Users/Shaik Faheem/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/
OneDrive/Desktop/intezam/ai6.py"
Input:  [3, '1', 2, '10', 'a', 5]
Sorted: ['1', 2, 3, 5, '10', 'a']

Input:  ['apple', '2', 10, '10.5', 'Banana', -1, '-2.5']
Sorted: ['-2.5', -1, '2', 10, '10.5', 'apple', 'Banana']

Input:  ['3.14', 2, '02', 'abc', 2.0]
Sorted: [2, '02', 2.0, '3.14', 'abc']
```

Observation:

Sorting raises TypeError because Python can't compare int and str; Copilot/Cursor can fix it by normalizing types (e.g., sorted(items, key=str) or converting all items to int) .

Task#3

• Write a Python snippet for file handling that opens a file but forgets to close it. Ask Copilot or Cursor AI to improve it using the best practice (e.g., with open() block)

Given bugged program to the ai:

```
ai6.py > ...
1   # buggy_file_handling.py
2
3   def read_file():
4       f = open("example.txt", "r")   # File opened but never closed!
5       data = f.read()
6       return data
7
8   print(read_file())
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

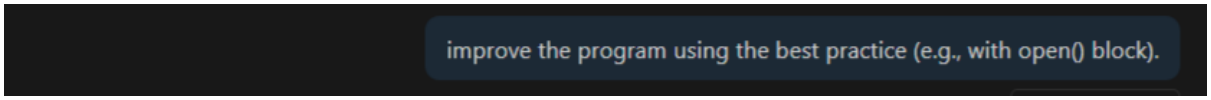
output of the bugged program:

```
Traceback (most recent call last):
  File "c:\Users\Shaik Faheem\OneDrive\Desktop\intezam\ai6.py", line 8, in <module>
    print(read_file())
          ^^^^^^^^^^^
  File "c:\Users\Shaik Faheem\OneDrive\Desktop\intezam\ai6.py", line 4, in read_file
    f = open("example.txt", "r")   # File opened but never closed!
        ^^^^^^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'example.txt'
PS C:\Users\Shaik Faheem\OneDrive\Desktop\intezam>
```

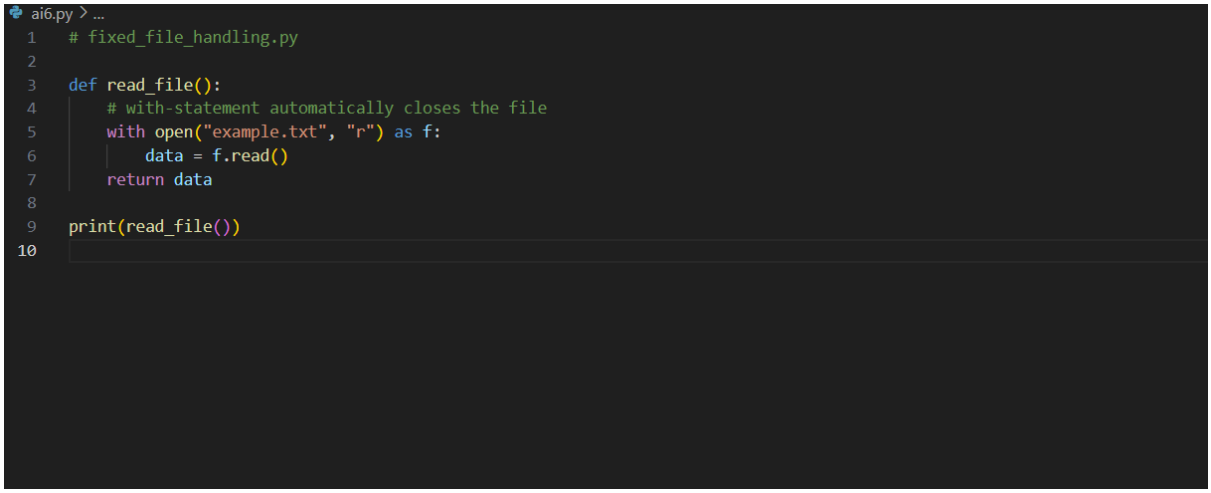
0 ⚠ 0 Ln 9, Col 1 Sp

given prompt to the ai:

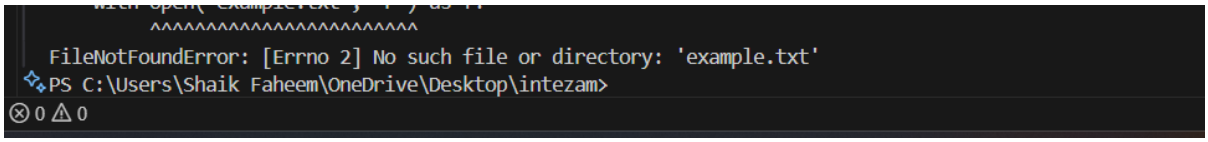
improve the program using the best practice (e.g., with open() block())



Given program by the ai:



Final output of the program:



Observation:

The file is opened but never closed, which may cause resource leaks; Copilot/Cursor can fix it by using a with open() .

Task#4

• Provide a piece of code with a ZeroDivisionError inside a loop. Ask AI to add error handling using try-except and continue execution safely

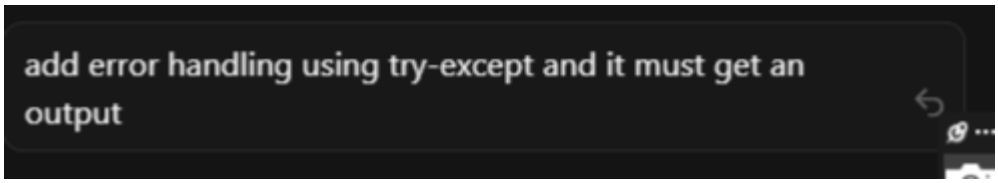
Given bugged program to the ai:

Output of the bugged program:

```
Traceback (most recent call last):
  File "c:\Users\Shaik Faheem\OneDrive\Desktop\intezam\ai6.py", line 6, in <module>
    result = 100 / n   # ⚠ZeroDivisionError when n = 0
             ~~~~^~~
ZeroDivisionError: division by zero
PS C:\Users\Shaik Faheem\OneDrive\Desktop\intezam>
0 ⚠ 0
```

Given prompt to the ai:

Add error handling using try-except and it must get an output



Given program by the ai:

```
ai6.py > ...
1   # fixed_division.py
2
3   numbers = [10, 5, 0, 2, 0, 8]
4
5   for n in numbers:
6       try:
7           result = 100 / n
8           print(f"100 / {n} = {result}")
9       except ZeroDivisionError:
10          print(f"Skipping division by zero for n = {n}")
11          continue
12
```

Final output of the program:

```
100 / 10 = 10.0
100 / 5 = 20.0
Skipping division by zero for n = 0
100 / 2 = 50.0
Skipping division by zero for n = 0
100 / 8 = 12.5
PS C:\Users\Shaik Faheem\OneDrive\Desktop\intezam>
```

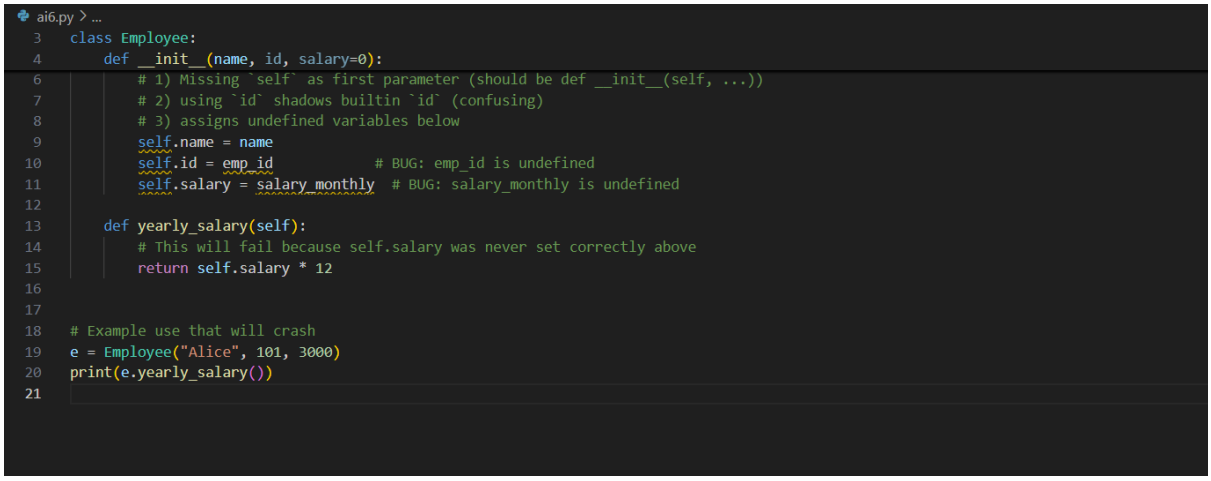
Observation:

The loop crashes with ZeroDivisionError when dividing by zero; Copilot/Cursor can fix it by wrapping the division in a try-except block and skipping or handling zero values to allow safe continuation.

Task#5

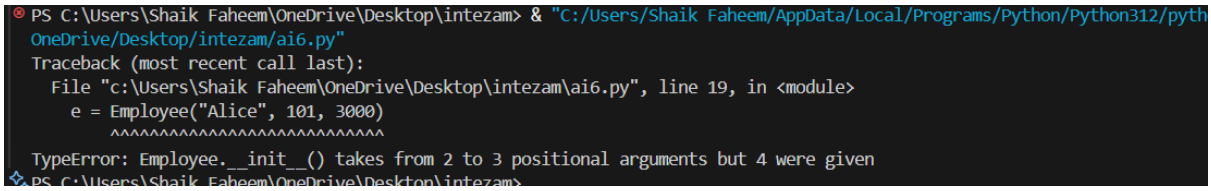
• Include a buggy class definition with incorrect __init__ parameters or attribute references. Ask AI to analyze and correct the constructor and attribute usage

Given bugged program to the ai:



```
ai6.py > ...
3   class Employee:
4       def __init__(name, id, salary=0):
6           # 1) Missing `self` as first parameter (should be def __init__(self, ...))
7           # 2) using `id` shadows builtin `id` (confusing)
8           # 3) assigns undefined variables below
9           self.name = name
10          self.id = emp_id            # BUG: emp_id is undefined
11          self.salary = salary_monthly  # BUG: salary_monthly is undefined
12
13      def yearly_salary(self):
14          # This will fail because self.salary was never set correctly above
15          return self.salary * 12
16
17
18  # Example use that will crash
19  e = Employee("Alice", 101, 3000)
20  print(e.yearly_salary())
21
```

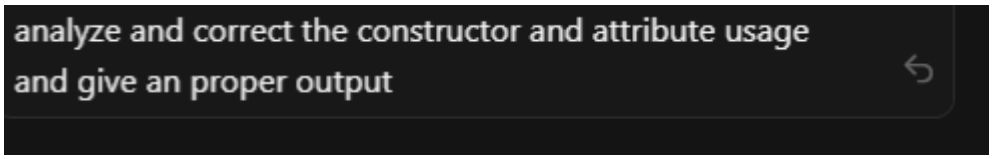
Output of the bugged program:



```
PS C:\Users\Shaik Faheem\OneDrive\Desktop\intezam> & "C:/Users/Shaik Faheem/AppData/Local/Programs/Python/Python312/pyth
OneDrive/Desktop/intezam/ai6.py"
Traceback (most recent call last):
  File "c:\Users\Shaik Faheem\OneDrive\Desktop\intezam\ai6.py", line 19, in <module>
    e = Employee("Alice", 101, 3000)
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: Employee.__init__() takes from 2 to 3 positional arguments but 4 were given
PS C:\Users\Shaik Faheem\OneDrive\Desktop\intezam>
```

Given prompt to the ai:

Analyze and correct the constructor and attribute usage and give an proper output



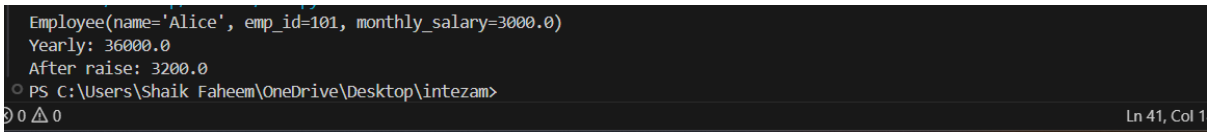
Given program by the ai:

```
class Employee:
    def give_raise(self, amount: float) -> None:
            raise ValueError("raise amount cannot be negative")
        self.monthly_salary += amount

    def __repr__(self) -> str:
        return (f"Employee(name={self.name!r}, emp_id={self.emp_id}, "
                f"monthly_salary={self.monthly_salary})")


# Example usage (works correctly)
if __name__ == "__main__":
    e = Employee("Alice", 101, 3000)
    print(e)                    # Employee(name='Alice', emp_id=101, monthly_salary=3000.0)
    print("Yearly:", e.yearly_salary())  # Yearly: 36000.0
    e.give_raise(200)
    print("After raise:", e.monthly_salary)  # After raise: 3200.0
```

Final output of the program:



Observation:

The class constructor uses wrong parameter/attribute references, causing NameError or AttributeError; Copilot/Cursor can fix it by aligning __init__ parameters with proper self attributes and updating method references consistently.