Lab 7: Error Debugging with AI (Week 4)

Subject: Ai assist coding
Experiment Title: Error Debugging with AI

Name: Sanjay Karupothula
Roll No: 2303A52337
BATCH: 34
Branch: BTech CSE
Lab: Week 4 – Error Debugging with AI
Date: 03-02-2026

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

• 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 – Runtime Error Due to Invalid Input Type

Buggy Code

```
num = input("Enter a number: ")
result = num + 10
print(result)
```

AI Explanation

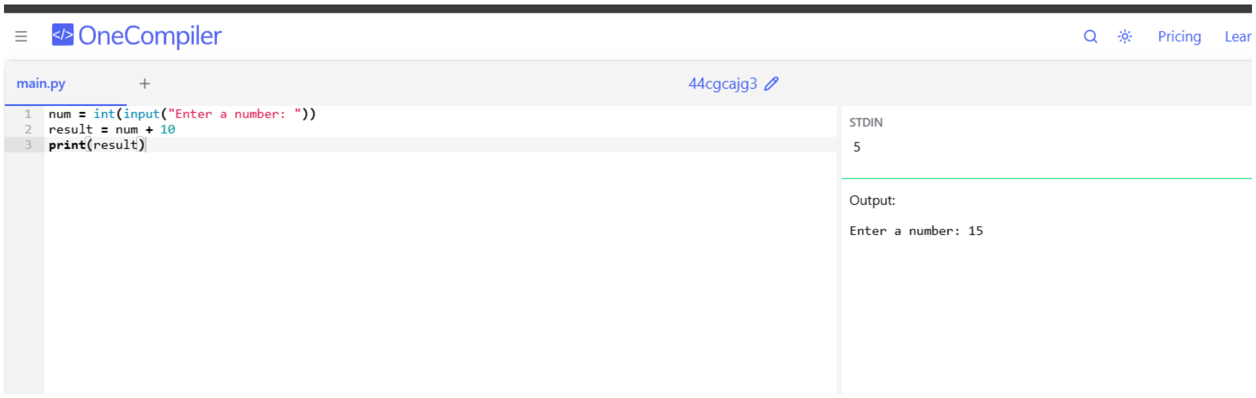
The input() function always returns a string. Adding an integer (10) to a string causes a TypeError. AI identifies that the user input must be converted to a numeric type such as int before performing arithmetic operations.

Corrected Code

```
num = int(input("Enter a number: "))
result = num + 10
print(result)
```

Output

```
Enter a number: 5
15
```



```
num = int(input("Enter a number: "))
result = num + 10
print(result)
```

STDIN

```
5
```

Output:

```
Enter a number: 15
```

Task 2 – Incorrect Function Return Value

Buggy Code

```
def square(n):
    result = n * n
```

AI Explanation

The function calculates the square but does not return the result. Without a return statement, Python returns None. AI suggests adding a return statement so the computed value is sent back to the caller.

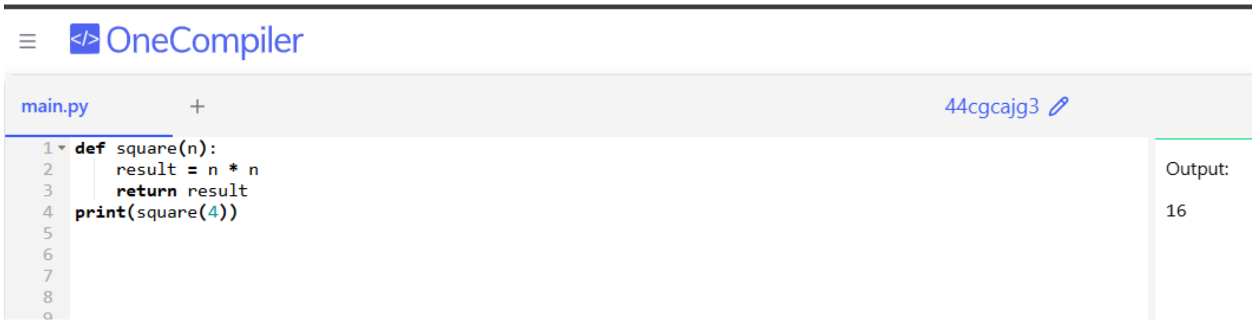
Corrected Code

```
def square(n):
    result = n * n
    return result
```

```
print(square(4))
```

Output

```
16
```



```
def square(n):
    result = n * n
    return result
print(square(4))
```

```
Output:

16
```

☑ Task 3 – IndexError in List Traversal

Buggy Code

```
numbers = [10, 20, 30]
for i in range(0, len(numbers)+1):
    print(numbers[i])
```

AI Explanation

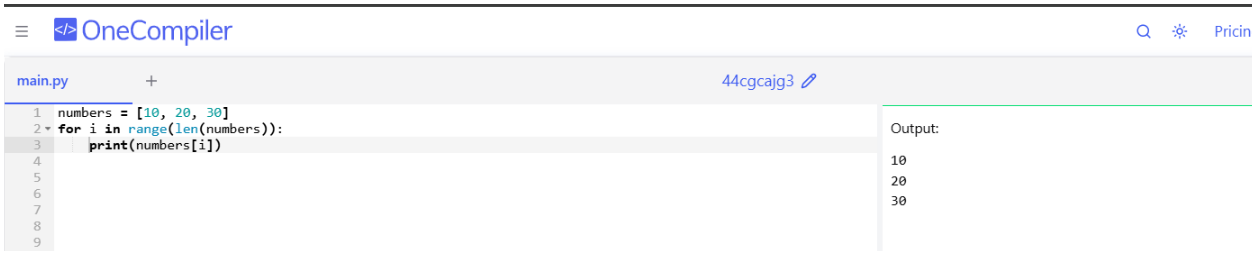
Using len(numbers)+1 makes the loop go one step beyond the last valid index, causing IndexError. AI fixes this by looping only up to len(numbers).

Corrected Code

```
numbers = [10, 20, 30]
for i in range(len(numbers)):
    print(numbers[i])
```

Output

```
10
20
30
```


OneCompiler
main.py
44cgcajg3

```
numbers = [10, 20, 30]
for i in range(len(numbers)):
    print(numbers[i])
```

Output:

```
10
20
30
```

☑ Task 4 – Uninitialized Variable Usage

Buggy Code

```
if True:
    pass
print(total)
```

AI Explanation

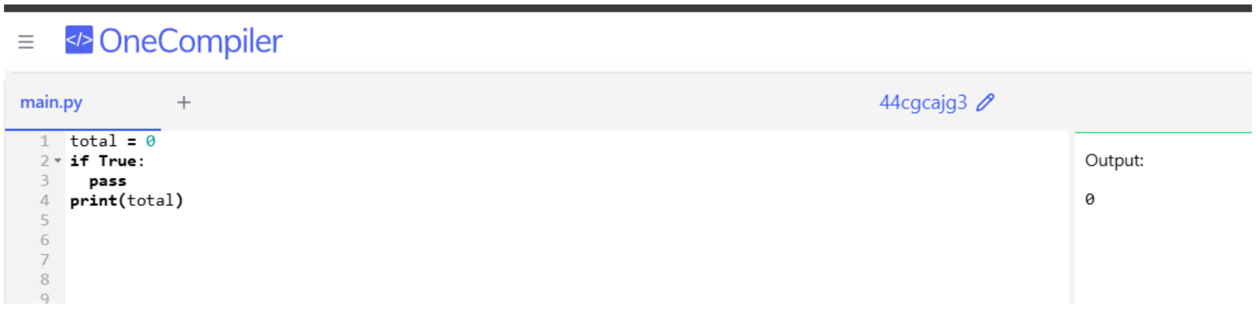
The variable total is used before being assigned any value, which causes a NameError. AI initializes the variable before it is used.

Corrected Code

```
total = 0
if True:
    pass
print(total)
```

Output

```
0
```


OneCompiler
main.py
44cgcajg3

```
total = 0
if True:
  pass
print(total)
```

Output:

```
0
```

☑ Task 5 – Logical Error in Student Grading System

Buggy Code

```
marks = 85
if marks >= 90:
    grade = "A"
elif marks >= 80:
    grade = "C"
else:
    grade = "B"
print(grade)
```

AI Explanation

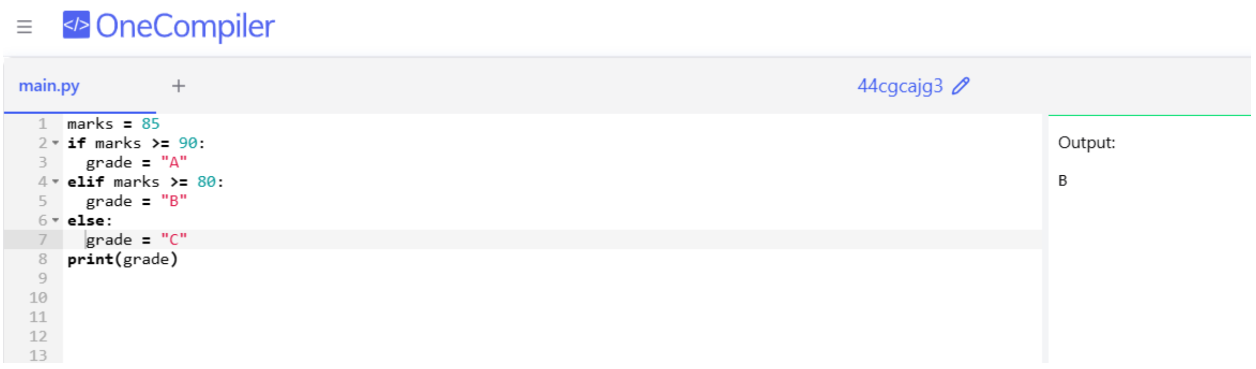
The grading logic is incorrect because marks between 80 and 89 should receive grade B, not C. AI rearranges the conditions to assign grades properly.

Corrected Code

```
marks = 85
if marks >= 90:
    grade = "A"
elif marks >= 80:
    grade = "B"
else:
    grade = "C"
print(grade)
```

Output

B



```
marks = 85
if marks >= 90:
  grade = "A"
elif marks >= 80:
  grade = "B"
else:
  grade = "C"
print(grade)
```

Output:

```
B
```

Conclusion

In this lab, different Python errors such as runtime errors, logical errors, index errors, and uninitialized variables were identified and corrected using AI assistance. AI helped explain each mistake and provided suitable fixes. This experiment improved debugging skills and increased confidence in writing reliable Python programs.