

AI Assisted Coding

Name : M.Harish

H.no : 2303A52387

Batch : 34

Course Code : 23CS002PC304

Assignment Number : 7.2

Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs

Task Description

Task 1 – Runtime Error Due to Invalid Input Type

- A Python program accepts user input and performs arithmetic operations. However, the program throws a runtime error because the input is treated as a string instead of a numeric type.

Example (Buggy Code):

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

- Task:

Use AI tools to identify the cause of the runtime error and modify the program so it executes correctly.

Expected Output -1:

- AI converts the input to the appropriate numeric type and eliminates the runtime error.

```

  num1_str = input("Enter the first number: ")
  num2_str = input("Enter the second number: ")

  # This will cause a TypeError if num1_str and num2_str are treated as strings and you try to do arithmetic.
  # For example, 5 + 6 results in '5', not 11.
  result_add_incorrect = num1_str + num2_str
  print(f"Incorrect addition result (string concatenation): {result_add_incorrect}")

  # result_multiply_incorrect = num1_str * num2_str # This would raise a TypeError immediately
  # print(f"Incorrect multiplication result: {result_multiply_incorrect}")

  Enter the first number: 5
  Enter the second number: 6
  Incorrect addition result (string concatenation): 56

```

The above code demonstrates the problem: `input()` always returns a string. When you try to perform arithmetic operations like addition or multiplication on strings, Python either concatenates them (`+`) or raises a `TypeError` (`*`).

To fix this, you need to explicitly convert the input strings to numeric types (like `int` for integers or `float` for decimal numbers) before performing arithmetic operations. Here's how to do it:

```

# Corrected program
# Get user input as strings
num1_str = input("Enter the first number: ")
num2_str = input("Enter the second number: ")

try:
    # Convert input strings to integers
    num1 = int(num1_str)
    num2 = int(num2_str)

    # Perform arithmetic operations
    sum_result = num1 + num2
    product_result = num1 * num2
    difference_result = num1 - num2
    quotient_result = num1 / num2 # Division might result in float

    print(f"Sum: {sum_result}")
    print(f"Product: {product_result}")
    print(f"Difference: {difference_result}")
    print(f"Quotient: {quotient_result}")

except ValueError:
    print("Invalid input. Please enter valid numbers.")
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")

Enter the first number: 1
Enter the second number: 2
Sum: 3
Product: 2
Difference: -1
Quotient: 0.5

```

Task 2 – Incorrect Function Return Value

A function is designed to calculate the square of a number, but it does not return the computed result properly.

Example (Buggy Code):

```
def square(n):
```

```
    result = n * n
```

Task:

Use AI assistance to analyze the function and ensure the correct value is returned.

Expected Output -2:

AI fixes the missing return statement and the function returns the correct output.

The screenshot shows a Google Colab interface with multiple tabs open. The active tab is 'Untitled1.ipynb'. The code in the cell is as follows:

```
# Incorrect function definition
def calculate_square_incorrect(number):
    result = number * number
    # Missing "return result" statement here

# Call the incorrect function
value = 7
incorrect_square = calculate_square_incorrect(value)

print(f"The square of {value} (incorrectly) is: {incorrect_square}")
print(f"Type of incorrect_square: {type(incorrect_square)}")
```

The output of this cell is:

```
The square of 7 (incorrectly) is: None
Type of incorrect_square: <class 'NoneType'>
```

A sidebar titled 'Gemini' provides an explanation of the issue:

M Incorrect Function Return Value A function is designed to calculate the square of a number, but it does not return the computed result properly.

I I understand! This is a common mistake when defining functions. A function might perform calculations, but if it doesn't explicitly `return` a value, it implicitly returns `None`. I'll show you an example of this problem and then provide the correct way to write the function.

The first code cell (8177f906) shows that the `calculate_square_incorrect` function returned `None`, as expected, because it lacked an explicit `return` statement. The output clearly states 'The square of 7 (incorrectly) is: None' and 'Type of incorrect_square: <class 'NoneType'>'.

In contrast, the second code cell (9999b321) demonstrates the corrected function, `calculate_square_correct`, which now explicitly returns the computed value. Its output confirms 'The square of 7 (correctly) is: 49' and 'Type of correct_square: <class 'int'>'. This successfully resolves the problem of the function not returning the computed result.

Both issues you raised have been fully addressed. I demonstrated and explained:

- 1. The issue of user input being treated as strings:** I showed how `input()` returns strings, leading to concatenation instead of arithmetic addition, and then provided a corrected version using `int()` to convert inputs to numbers.
- 2. The problem of functions not returning values:** I illustrated how a function implicitly returns `None` without an explicit `return` statement, and then corrected the function to properly return the computed result.

All code has been executed, and the outputs have been explained. If you have any new questions or would like to explore other Python concepts, please let me know!

What can I help you build? Gemini 2.5 Flash +

Variables Terminal 22°C Sunny 9:55 AM ENG IN 09:55 03-02-2026 Python 3

Task 3 – IndexError in List Traversal

A Python program iterates over a list using incorrect index limits, causing an `IndexError`.

Example (Buggy Code):

```
numbers = [10, 20, 30]
```

```
for i in range(0, len(numbers)+1):
```

```
    print(numbers[i])
```

Task:

Use AI to identify the incorrect loop boundary and correct the iteration logic.

Expected Output -3:

AI fixes the loop condition and prevents out-of-range list access.

The screenshot shows a Google Colab interface with multiple tabs open. The active tab is 'Untitled1.ipynb'. The code in the cell is as follows:

```

my_list = ['apple', 'banana', 'cherry']

# Incorrect iteration: trying to access an index equal to the length of the list
# Python lists are 0-indexed, so valid indices for a list of length N are 0 to N-1.
# len(my_list) is 3, so trying to access index 3 will cause an IndexError.
# print("Attempting incorrect list traversal:")
for i in range(len(my_list) + 1):
    try:
        print(f"Element at index {i}: {my_list[i]}")
    except IndexError as e:
        print(f"Caught an error: {e} at index {i}")

... Attempting incorrect list traversal:
Element at index 0: apple
Element at index 1: banana
Element at index 2: cherry
Caught an error: list index out of range at index 3

```

Below the code, a note explains the error:

You can see that the `IndexError: list index out of range` occurs when the loop tries to access `my_list[3]`. Python lists use 0-based indexing, meaning the first element is at index 0, the second at index 1, and so on. For a list of length 3, the valid indices range from 0 to N-1.

The `range(len(my_list) + 1)` in the incorrect example generates numbers from 0 up to `len(my_list)` (inclusive of `len(my_list)`, but exclusive of `len(my_list) + 1`). This causes it to attempt to access an index that doesn't exist.

Here's the corrected version, using `range(len(my_list))` which correctly generates indices from 0 to N-1:

```

my_list = ['apple', 'banana', 'cherry']

# Correct iteration using range(len(list)):
print("Performing correct list traversal:")
for i in range(len(my_list)):
    print(f"Element at index {i}: {my_list[i]}")

# Even better: direct iteration over elements
print("Performing direct iteration over elements:")
for item in my_list:
    print(f"Element: {item}")

# Or using enumerate for both index and element
print("Performing iteration with enumerate:")
for index, item in enumerate(my_list):
    print(f"Element at index {index}: {item}")

Performing correct list traversal:
Element at index 0: apple
Element at index 1: banana
Element at index 2: cherry

Performing direct iteration over elements:
Element: apple
Element: banana
Element: cherry

Performing iteration with enumerate:
Element at index 0: apple
Element at index 1: banana
Element at index 2: cherry

```

Task 4 – Uninitialized Variable Usage

A program uses a variable in a calculation before assigning it any value.

Example (Buggy Code):

if True:

pass

print(total)

Task:

Use AI tools to detect the uninitialized variable and correct the program.

Expected Output -4:

AI initializes the variable correctly before it is used

Task Description

Task – Logical Error in Student Grading System

A grading program assigns incorrect grades due to improper conditional

logic.

Example (Buggy Code):

```
marks = 85
```

```
if marks >= 90:
```

```
    grade = "A"
```

```
elif marks >= 80:
```

```
    grade = "C"
```

```
else:
```

```
    grade = "B"
```

```
print(grade)
```

Task:

Use AI to analyze the grading conditions and correct the logical flow.

Expected Output -5:

AI corrects the conditional logic so grades are assigned accurately

The screenshot shows a Google Colab interface with several tabs open:

- AI Assisted Debugging
- AIAC-LAB6.5.ipynb - Colab
- Lab Assignment 7.2
- Untitled1.ipynb - Colab
- New tab

The main notebook window displays the following Python code:

```
# Incorrect: Using 'x' before it's defined
def calculate_sum_incorrect():
    y = 10
    result = x + y # This line will cause a NameError
    print(f"The sum (incorrectly): {result}")

try:
    calculate_sum_incorrect()
except NameError as e:
    print(f"Caught an error: {e}")

# Another example: trying to print an uninitialized variable
try:
    print("Uninitialized variable: (uninitialized_var)")
except NameError as e:
    print(f"Caught another error: {e}")

... Caught an error: name 'x' is not defined
... Caught another error: name 'uninitialized_var' is not defined
```

A sidebar titled "Gemini" provides feedback:

- Uninitialized Variable Usage**: A program uses a variable in a calculation before assigning it any value. Expected Output -4.
- You're encountering a `NameError`, which happens when you try to use a variable that hasn't been assigned a value yet. I'll provide an example that causes this error, explain it, and then show you the correct way to initialize and use variables.

The code editor shows a corrected version of the script:

```
# Corrected: Initializing 'x' before use
def calculate_sum_correct():
    x = -4 # 'x' is now initialized
    y = 10
    result = x + y
    print(f"The sum (correctly): {result}")

calculate_sum_correct()

# Correcting initializing 'my_variable' before use
my_variable = "Hello, World!"
print(f"Initialized variable: {my_variable}")

The sum (correctly): -4
Initialized variable: Hello, World!
```

The bottom status bar indicates the environment is "Python 3".