Name: Achi Vaishnavi

Hall Ticket:2303A51554

Batch:29

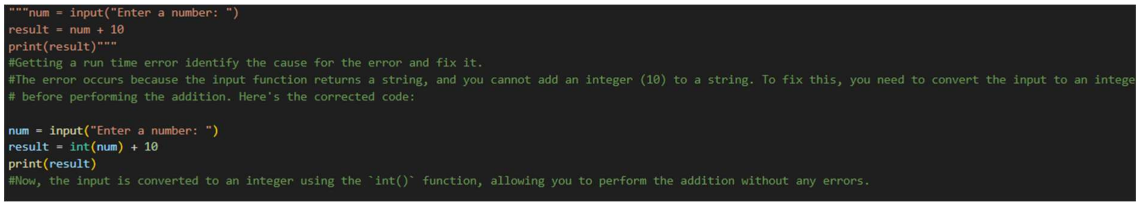
Assessment: 7.2

Task 1 – Runtime Error Due to Invalid Input Type

Prompt:

Getting a run time error identify the cause for the error and fix it.

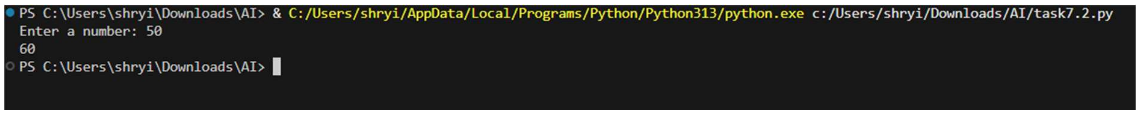
Code :



```
"""num = input("Enter a number: ")
result = num + 10
print(result)"""
#Getting a run time error identify the cause for the error and fix it.
#The error occurs because the input function returns a string, and you cannot add an integer (10) to a string. To fix this, you need to convert the input to an intege
# before performing the addition. Here's the corrected code:

num = input("Enter a number: ")
result = int(num) + 10
print(result)
#Now, the input is converted to an integer using the `int()` function, allowing you to perform the addition without any errors.
```

Output:



```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe c:/Users/shryi/Downloads/AI/task7.2.py
Enter a number: 50
60
PS C:\Users\shryi\Downloads\AI> 
```

Justification :

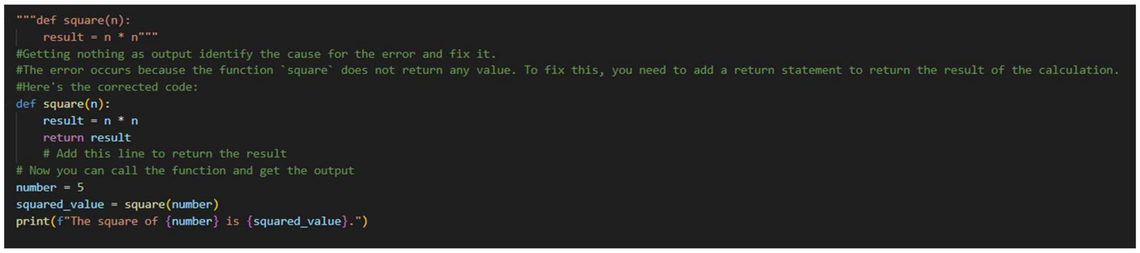
The error occurred because the value returned by input() is always a string, and arithmetic operations cannot be performed directly on strings. When the program tried to add a number to this string, Python raised a TypeError. By converting the input into an integer using int(), the program was able to execute the calculation correctly. This task helped me understand how user input is processed in Python and why explicit type conversion is necessary for numerical operations.

Task 2 : Incorrect Function Return Value

Prompt:

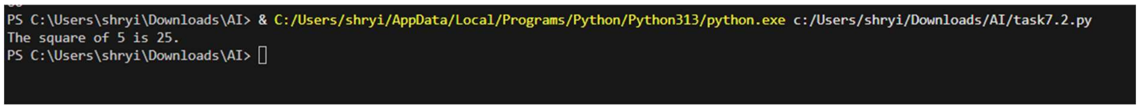
Not getting an output identify the cause for the error and fix it.

Code :



```python
"""def square(n):
    result = n * n"""
#Getting nothing as output identify the cause for the error and fix it.
#The error occurs because the function `square` does not return any value. To fix this, you need to add a return statement to return the result of the calculation.
#Here's the corrected code:
def square(n):
    result = n * n
    return result
    # Add this line to return the result
# Now you can call the function and get the output
number = 5
squared_value = square(number)
print(f"The square of {number} is {squared_value}.")
```

Output :



```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe c:/Users/shryi/Downloads/AI/task7.2.py
The square of 5 is 25.
PS C:\Users\shryi\Downloads\AI> 
```

Justification :

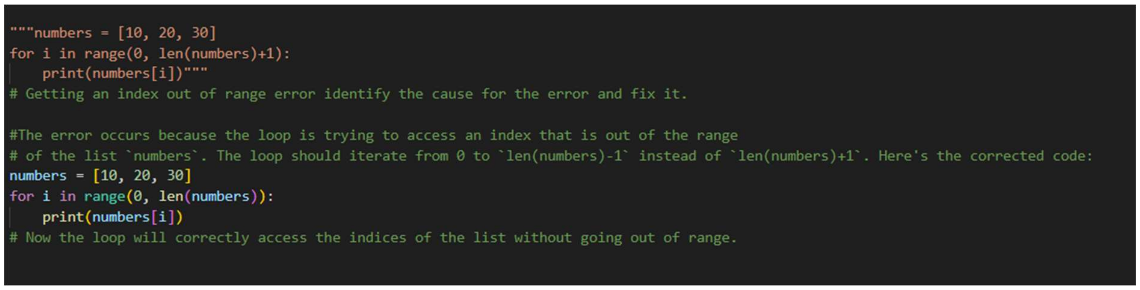
The issue occurred because the function calculated the result internally but did not send it back to the caller. Without a return statement, Python automatically returns None, which is why no meaningful output was displayed. By adding a return statement, the function was able to provide the computed value to the print statement. This task strengthened my understanding of how data flows from a function and why returning values is essential for producing output.

Task 3: IndexError in List Traversal

Prompt :

Getting an index out of range error identify the cause for the error and fix it.

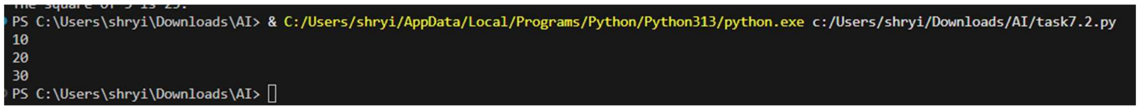
Code :



```python
"""numbers = [10, 20, 30]
for i in range(0, len(numbers)+1):
    print(numbers[i])"""
# Getting an index out of range error identify the cause for the error and fix it.

#The error occurs because the loop is trying to access an index that is out of the range
# of the list `numbers`. The loop should iterate from 0 to `len(numbers)-1` instead of `len(numbers)+1`. Here's the corrected code:
numbers = [10, 20, 30]
for i in range(0, len(numbers)):
    print(numbers[i])
# Now the loop will correctly access the indices of the list without going out of range.
```

Output :



```
PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe c:/Users/shryi/Downloads/AI/task7.2.py
10
20
30
PS C:\Users\shryi\Downloads\AI> 
```

Justification :

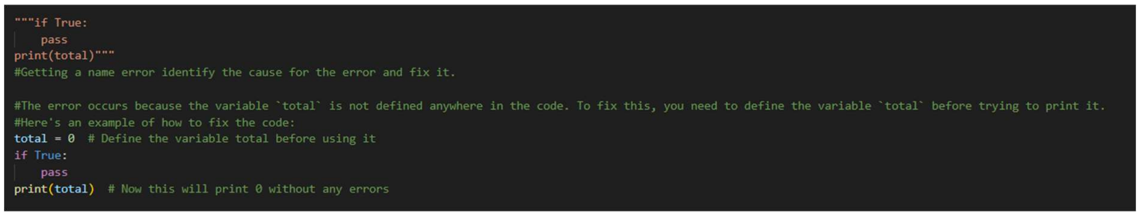
The IndexError occurred because the loop tried to access a position outside the list's valid range. Since Python lists start indexing from zero, the highest accessible index is always one less than the list length. Updating the loop to use the correct range prevented out-of-bounds access. This task helped me understand proper loop limits and reinforced safe list traversal practices.

Task 4 :Uninitialized Variable Usage

Prompt :

Getting a name error identify the cause for the error and fix it.

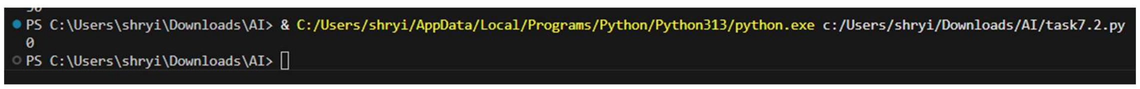
Code:



```
"""if True:
    pass
print(total)"""
#Getting a name error identify the cause for the error and fix it.

#The error occurs because the variable `total` is not defined anywhere in the code. To fix this, you need to define the variable `total` before trying to print it.
#Here's an example of how to fix the code:
total = 0  # Define the variable total before using it
if True:
    pass
print(total)  # Now this will print 0 without any errors
```

Output:



```
● PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe c:/Users/shryi/Downloads/AI/task7.2.py
0
○ PS C:\Users\shryi\Downloads\AI>
```

Justification :

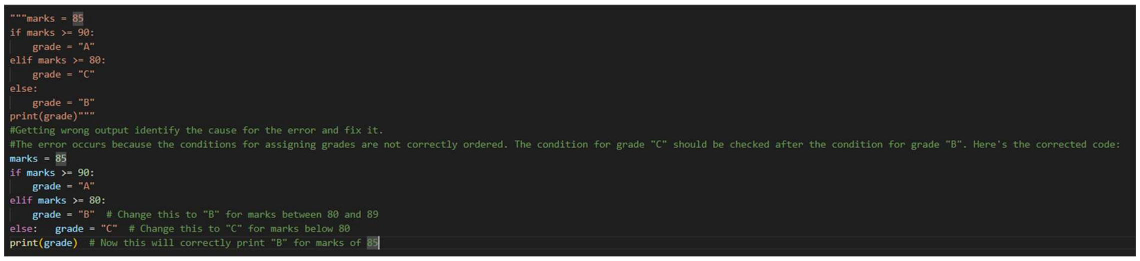
The error occurred because the program attempted to use a variable that had not yet been created. Python cannot recognize variables unless they are assigned a value beforehand, which resulted in a NameError. By defining the variable before accessing it, the program ran successfully. This task improved my understanding of variable initialization and highlighted the importance of assigning values before using variables in a program.

Task 5: Logical Error in Student Grading System

Prompt:

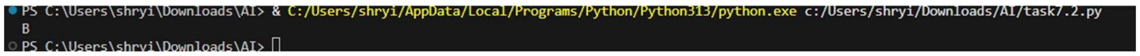
Getting wrong output identify the cause for the error and fix it.

Code :



```
"""marks = 85
if marks >= 90:
    grade = "A"
elif marks >= 80:
    grade = "C"
else:
    grade = "B"
print(grade)"""
#Getting wrong output identify the cause for the error and fix it.
#The error occurs because the conditions for assigning grades are not correctly ordered. The condition for grade "C" should be checked after the condition for grade "B". Here's the corrected code:
marks = 85
if marks >= 90:
    grade = "A"
elif marks >= 80:
    grade = "B"  # Change this to "B" for marks between 80 and 89
else:   grade = "C"  # Change this to "C" for marks below 80
print(grade)  # Now this will correctly print "B" for marks of 85
```

Output:



```
● PS C:\Users\shryi\Downloads\AI> & C:/Users/shryi/AppData/Local/Programs/Python/Python313/python.exe c:/Users/shryi/Downloads/AI/task7.2.py
B
○ PS C:\Users\shryi\Downloads\AI>
```

Justification :

The program produced incorrect results because the general condition was checked before the specific grading condition, causing the correct grade to be skipped. Python evaluates if-elif statements from top to bottom, so placing higher-score conditions first is necessary. Rearranging the conditions fixed the logic and produced accurate grades. This task helped me understand how condition order affects program flow and output accuracy.