```
**2303A52191**
```

Task 1 – Runtime Error Due to Invalid Input Type

Double-click (or enter) to edit

Task 1: Runtime Error Due to Invalid Input Type

Problem: 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.

```
# Buggy Code for Task 1
 num = input("Enter a number: ")
 result = num + 10
 print(result)

# To demonstrate the error, uncomment the lines above and run this cell.
# You will get a TypeError because you cannot add an integer to a string.
```

```
  File "/tmp/ipython-input-3416380673.py", line 2
    num = input("Enter a number: ")
    ^
IndentationError: unexpected indent
```

Next steps: Explain error

Solution for Task 1: Convert the input to the appropriate numeric type (e.g., `int` or `float`) and handle potential `ValueError` if the input is not a valid number.

```
# Corrected Code for Task 1
num_str = input("Enter a number: ")
try:
    num_int = int(num_str)  # Convert the input string to an integer
    result = num_int * 5
    print(f"The result is: {result}")
except ValueError:
    print("Invalid input. Please enter a valid integer.")

print("\n--- End of Task 1 ---\n")
```

```
Enter a number: 4
The result is: 20

--- End of Task 1 ---
```

Task 2: Incorrect Function Return Value

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

```
# Buggy Code for Task 2
def square(n):
    result = n * n
# To demonstrate the error, call the function and try to print its return value:
# print(square(5)) # This will print 'None'
# The function implicitly returns None because there is no explicit return statement.
```

Solution for Task 2: Add a `return` statement to the function to ensure the calculated result is returned.

```
# Corrected Code for Task 2
def square(n):
    result = n * n
    return result # Added return statement

# Test the corrected function
num_to_square = 7
squared_num = square(num_to_square)
print(f"The square of {num_to_square} is: {squared_num}")

print("\n--- End of Task 2 ---\n")
```

```
The square of 7 is: 49

--- End of Task 2 ---
```

Task 3: IndexError in List Traversal

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

```
# Buggy Code for Task 3
numbers = [10, 20, 30]
for i in range(0, len(numbers) + 1):
    # This loop will cause an IndexError because len(numbers) is 3,
    # and range(0, 4) will try to access index 3, which is out of bounds for a 0-indexed list of length 3.
    print(numbers[i])
```

```
10
20
30
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipython-input-1867278540.py in <cell line: 0>()
      4     # This loop will cause an IndexError because len(numbers) is 3,
      5     # and range(0, 4) will try to access index 3, which is out of bounds for a 0-indexed list of
length 3.
----> 6     print(numbers[i])

IndexError: list index out of range
```

Next steps: Explain error

Solution for Task 3: Correct the loop's range to iterate only over valid indices (from 0 up to `len(list) - 1`). Using `range(len(list))` or iterating directly over the elements is recommended.

```
# Corrected Code for Task 3
numbers = [10, 20, 30]

# Method 1: Using range(len(numbers)) which goes from 0 to len(numbers)-1
print("Method 1: Corrected loop using range(len(numbers))")
for i in range(len(numbers)):
    print(numbers[i])

# Method 2: Directly iterating over elements (more Pythonic)
print("\nMethod 2: Directly iterating over elements")
for num in numbers:
    print(num)

print("\n--- End of Task 3 ---\n")
```

```
Method 1: Corrected loop using range(len(numbers))
10
20
30

Method 2: Directly iterating over elements
```

```
10
20
30

--- End of Task 3 ---
```

Task 4: Uninitialized Variable Usage

Problem: A Python program attempts to use a variable in a calculation before it has been assigned any value, leading to a `NameError`.

```
# Buggy Code for Task 4
def calculate_sum():
    # 'total' is used before it's assigned a value in some paths
    if False:
        total = 10
    result = total + 5
    print(result)

# To demonstrate the error, uncomment the line below and run this cell.
# This will cause a NameError because 'total' is not initialized.
# calculate_sum()
```

Solution for Task 4: Ensure all variables are initialized with a default value before being used in calculations, especially in conditional blocks where assignment might not always occur.

```
# Corrected Code for Task 4
def calculate_sum_corrected():
    total = 0  # Initialize 'total' with a default value
    if False:
        total = 10
    result = total + 5
    print(f"The result is: {result}")

# Test the corrected function
calculate_sum_corrected()

print("\n--- End of Task 4 ---\n")
```

```
The result is: 5

--- End of Task 4 ---
```

Task 5 – Logical Error in Student Grading System

A grading program assigns incorrect grades due to improper conditional logic.

```
# Buggy Code for Task 5
def assign_grade(score):
    if score >= 90:
        return "A"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "F"
    elif score >= 50:
        return "B"
    elif score < 50:
        return "D"
    else:
        return "Invalid Score"

# To demonstrate the error, uncomment and run:
# print(f"Score 85: {assign_grade(85)}") # Expected B, might be A if logic is wrong
# print(f"Score 65: {assign_grade(65)}") # Expected C, might be B if logic is wrong
# print(f"Score 45: {assign_grade(45)}") # Expected F
# The current logic will correctly assign grades in many cases, but if the order of conditions were wrong (e
# The current logic will correctly assign grades in many cases, but if the order of conditions were wrong (s
```

```
# Corrected Code for Task 5
def assign_grade_corrected(score):
    if score >= 90:
        return "A"
    elif score >= 80: # Corrected range for B
        return "B"
    elif score >= 70: # Corrected range for C
        return "C"
    elif score >= 60: # Corrected range for D
        return "D"
    else: # Any score below 60 is F
        return "F"

# Test the corrected function
print(f"Score 95: {assign_grade_corrected(95)}") # Expected A
print(f"Score 85: {assign_grade_corrected(85)}") # Expected B
print(f"Score 75: {assign_grade_corrected(75)}") # Expected C
print(f"Score 65: {assign_grade_corrected(65)}") # Expected D
print(f"Score 55: {assign_grade_corrected(55)}") # Expected F
print(f"Score 45: {assign_grade_corrected(45)}") # Expected F
print("\n--- End of Task 5 ---\n")
```

```
Score 95: A
Score 85: B
Score 75: C
Score 65: D
Score 55: F
Score 45: F

--- End of Task 5 ---
```