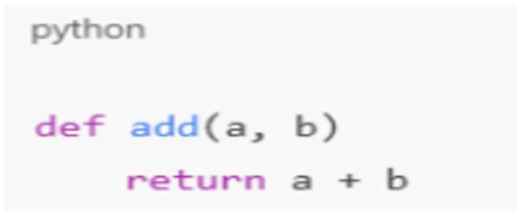
Name: M.sivateja

Hall Ticket: 2303A52145

Batch: 41

Task 1: Fixing Syntax Errors

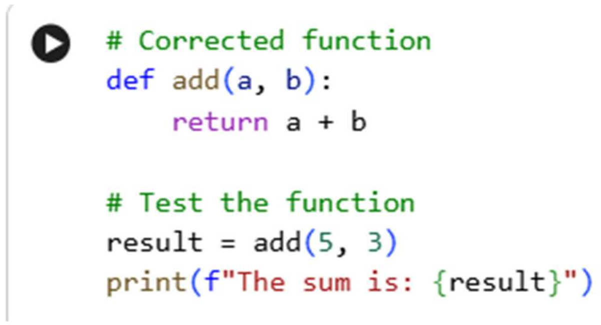


```python
def add(a, b)
    return a + b
```

Prompt:

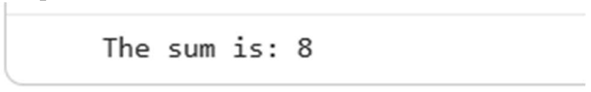
def add(a,b) return a+b
detect the syntax error in the above code, correct the function definition and explain the syntax issue and give correct output

Code:



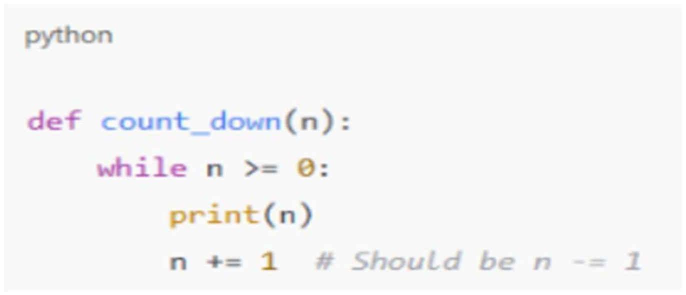
```python
# Corrected function
def add(a, b):
    return a + b

# Test the function
result = add(5, 3)
print(f"The sum is: {result}")
```

Output:



```
The sum is: 8
```

Task 2: Debugging Logic Errors in Loops

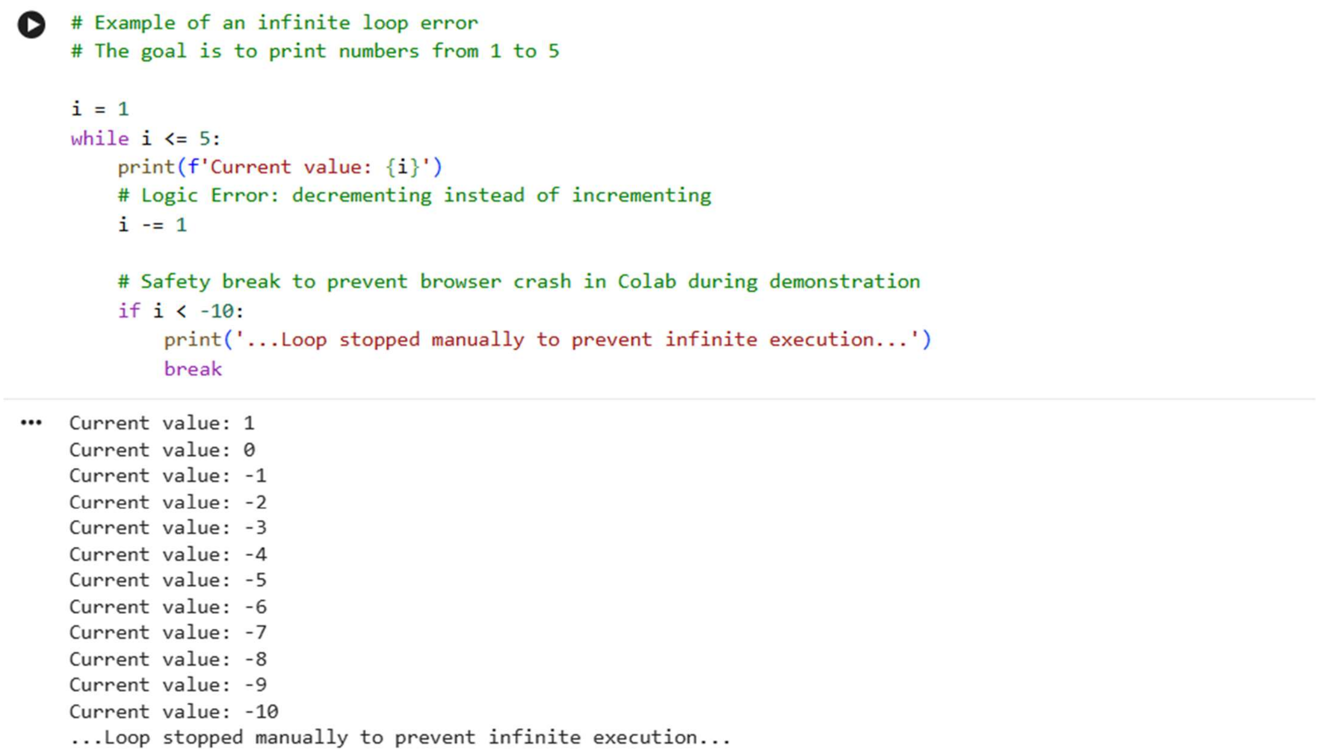


```python
def count_down(n):
    while n >= 0:
        print(n)
        n += 1  # Should be n -= 1
```

Prompt:

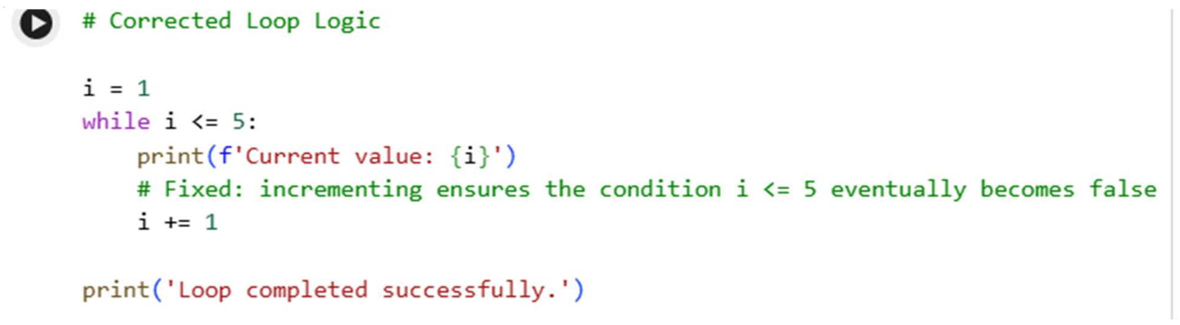
fix the error in the code by Providing a loop with an increment or decrement error, identify the cause of infinite iteration and fix the loop logic
• Analyze the corrected loop behavior and explain the logic error

Code:



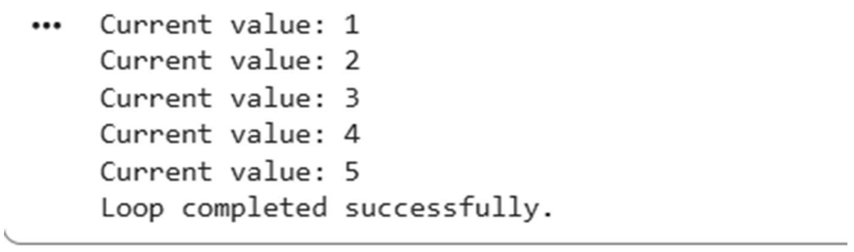
```python
# Example of an infinite loop error
# The goal is to print numbers from 1 to 5

i = 1
while i <= 5:
    print(f'Current value: {i}')
    # Logic Error: decrementing instead of incrementing
    i -= 1

    # Safety break to prevent browser crash in Colab during demonstration
    if i < -10:
        print('...Loop stopped manually to prevent infinite execution...')
        break
```

```
Current value: 1
Current value: 0
Current value: -1
Current value: -2
Current value: -3
Current value: -4
Current value: -5
Current value: -6
Current value: -7
Current value: -8
Current value: -9
Current value: -10
...Loop stopped manually to prevent infinite execution...
```



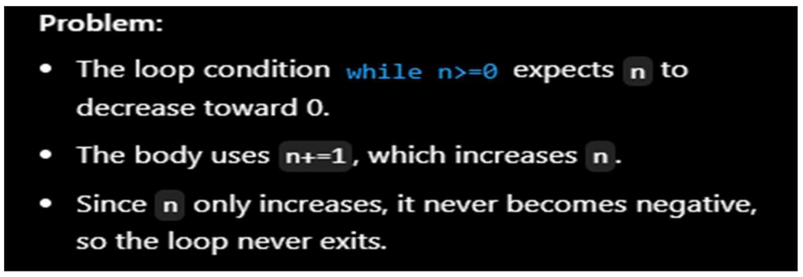
```python
# Corrected Loop Logic

i = 1
while i <= 5:
    print(f'Current value: {i}')
    # Fixed: incrementing ensures the condition i <= 5 eventually becomes false
    i += 1

print('Loop completed successfully.')
```

Output:



```
Current value: 1
Current value: 2
Current value: 3
Current value: 4
Current value: 5
Loop completed successfully.
```

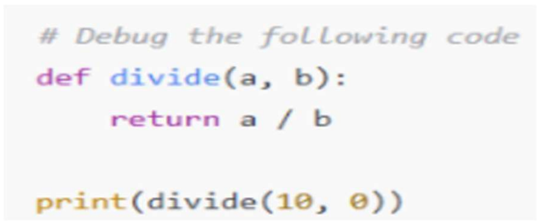
Explanation:



Problem:

- The loop condition `while n>=0` expects `n` to decrease toward 0.
- The body uses `n+=1`, which increases `n`.
- Since `n` only increases, it never becomes negative, so the loop never exits.

Task 3: Handling Runtime Errors (Division by Zero)

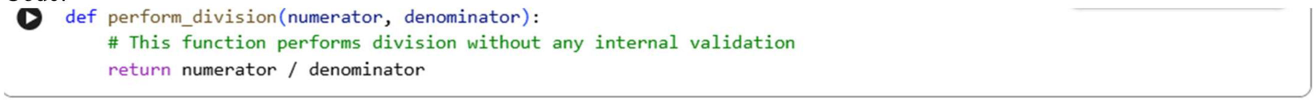


```python
# Debug the following code
def divide(a, b):
    return a / b

print(divide(10, 0))
```

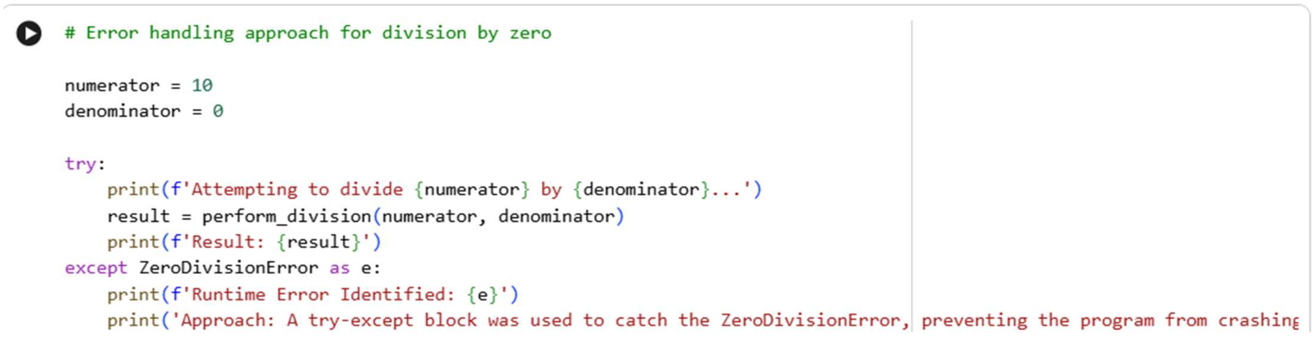
Prompt:

Debug the following code.add try-except blocks to handle division by zero error.Provide a function that performs division without validation.give error handlingapproach and identify the runtime error

Code:

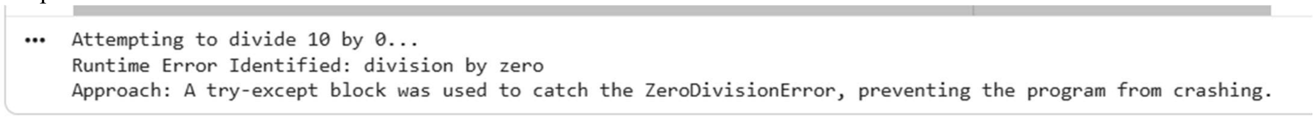


```python
def perform_division(numerator, denominator):
    # This function performs division without any internal validation
    return numerator / denominator
```



```python
# Error handling approach for division by zero

numerator = 10
denominator = 0

try:
    print(f'Attempting to divide {numerator} by {denominator}...')
    result = perform_division(numerator, denominator)
    print(f'Result: {result}')
except ZeroDivisionError as e:
    print(f'Runtime Error Identified: {e}')
    print('Approach: A try-except block was used to catch the ZeroDivisionError, preventing the program from crashing
```

Output:

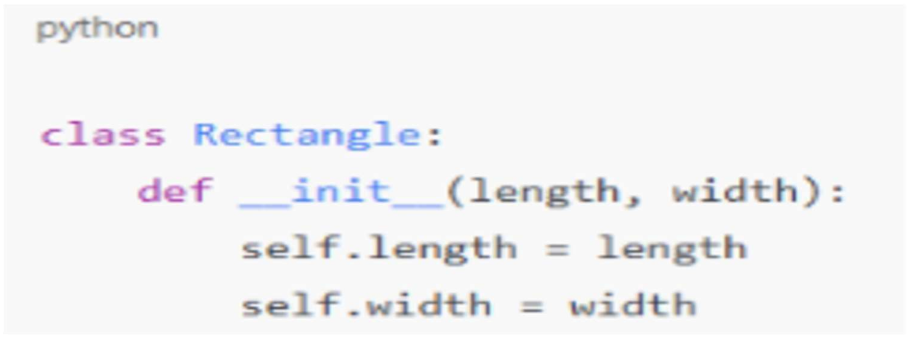


```
Attempting to divide 10 by 0...
Runtime Error Identified: division by zero
Approach: A try-except block was used to catch the ZeroDivisionError, preventing the program from crashing.
```

Explanation:

The error happens because division by zero is not permitted in Python, which raises a ZeroDivisionError when divide(10, 0) is called. Using a try–except block prevents the program from crashing and displays a clear error message instead.

Task 4: Debugging Class Definition Errors

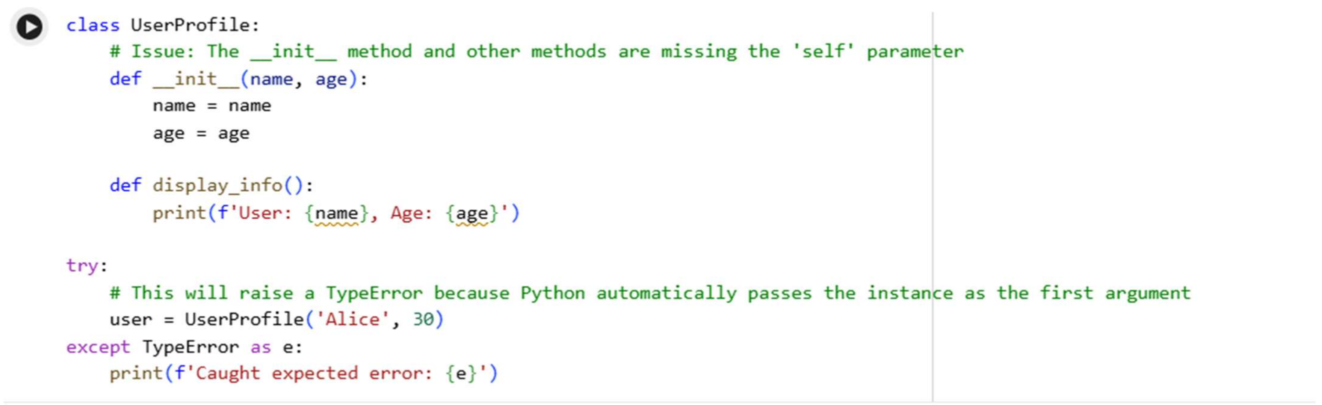


```python
python

class Rectangle:
    def __init__(length, width):
        self.length = length
        self.width = width
```

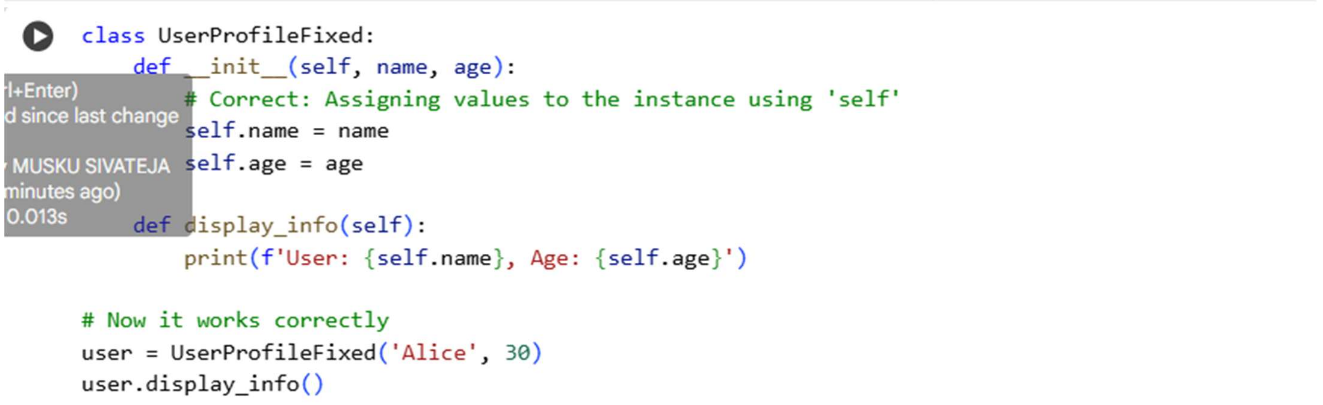
Prompt:

Provide a class definition with missing self-parameter. identify the issue in the __init__() method.correct the class definition.Understand why self is required and explain it

Code:

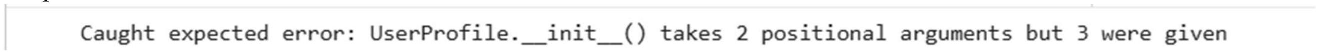


```python
class UserProfile:
    # Issue: The __init__ method and other methods are missing the 'self' parameter
    def __init__(name, age):
        name = name
        age = age

    def display_info():
        print(f'User: {name}, Age: {age}')

try:
    # This will raise a TypeError because Python automatically passes the instance as the first argument
    user = UserProfile('Alice', 30)
except TypeError as e:
    print(f'Caught expected error: {e}')
```

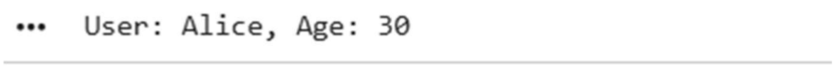


```python
class UserProfileFixed:
    def __init__(self, name, age):
        # Correct: Assigning values to the instance using 'self'
        self.name = name
        self.age = age

    def display_info(self):
        print(f'User: {self.name}, Age: {self.age}')

# Now it works correctly
user = UserProfileFixed('Alice', 30)
user.display_info()
```

Output:



```
Caught expected error: UserProfile.__init__() takes 2 positional arguments but 3 were given
```

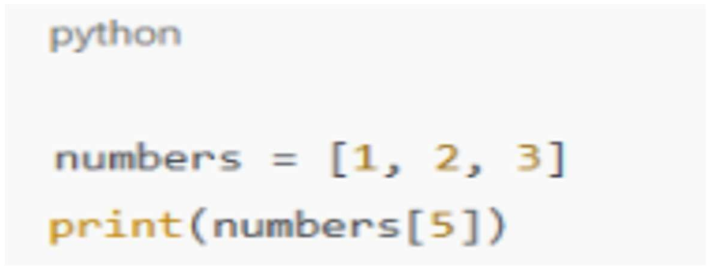


```
User: Alice, Age: 30
```

Explanation:

The problem occurred because the **init**() method did not include the self parameter, so the object could not store its values properly. Adding self allows length and width to be saved as attributes and lets the class correctly initialize and access each Rectangle object.

Task 5: Resolving Index Errors in Lists

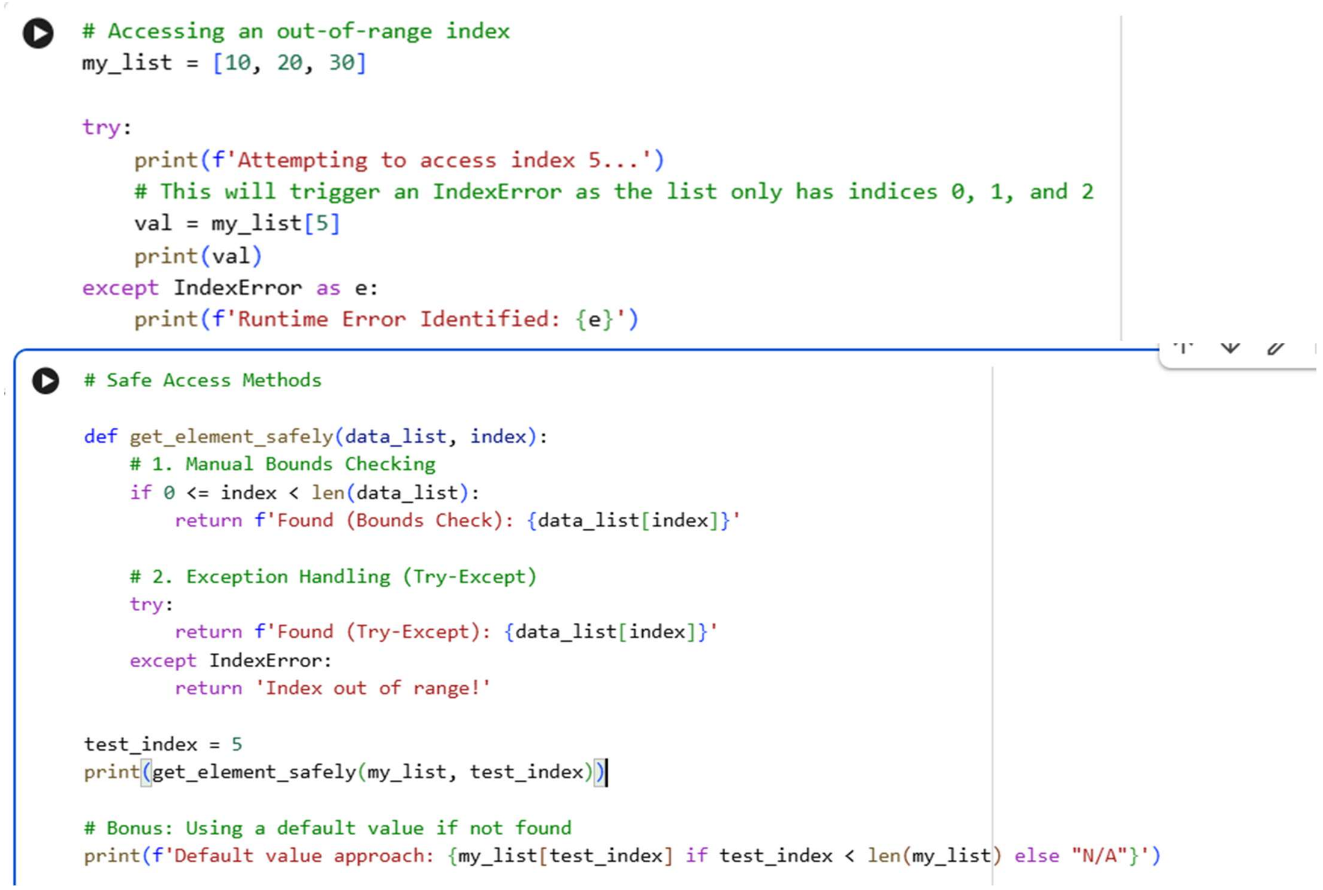


```python
numbers = [1, 2, 3]
print(numbers[5])
```

Prompt:

Provide code that accesses an out-of-range list index.identify the Index Error.suggest safe access methods.Apply bounds checking or exception handling.explain the changes made

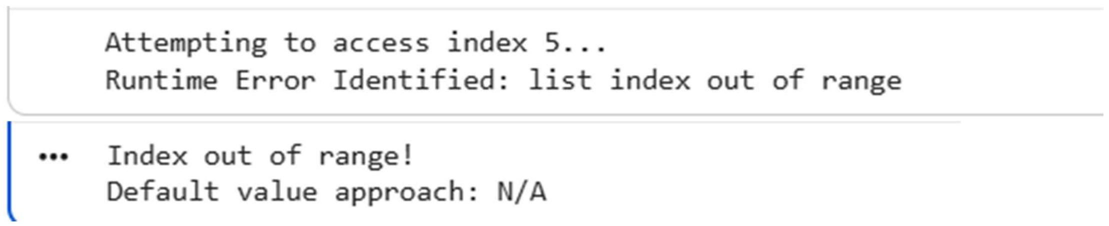
Code:



```python
# Accessing an out-of-range index
my_list = [10, 20, 30]

try:
    print(f'Attempting to access index 5...')
    # This will trigger an IndexError as the list only has indices 0, 1, and 2
    val = my_list[5]
    print(val)
except IndexError as e:
    print(f'Runtime Error Identified: {e}')
```

```python
# Safe Access Methods

def get_element_safely(data_list, index):
    # 1. Manual Bounds Checking
    if 0 <= index < len(data_list):
        return f'Found (Bounds Check): {data_list[index]}'

    # 2. Exception Handling (Try-Except)
    try:
        return f'Found (Try-Except): {data_list[index]}'
    except IndexError:
        return 'Index out of range!'

test_index = 5
print(get_element_safely(my_list, test_index))

# Bonus: Using a default value if not found
print(f'Default value approach: {my_list[test_index] if test_index < len(my_list) else "N/A"}')
```

OUTPUT:



```
Attempting to access index 5...
Runtime Error Identified: list index out of range
```

```
Index out of range!
Default value approach: N/A
```

Explanation:

The error happens because the code tries to access an index that is not present in a list with only three items, which raises an IndexError.
Using a try–except block prevents the program from crashing and displays a helpful message instead.