


Corrected Code:

```

C:\Users\akshi> cd C:\OneDrive\Desktop & python ass - 7.4.py
1  def factorial(n):
2      if n == 0 or n == 1:
3          return 1
4      return n * factorial(n - 1)
5
6  n = int(input("Enter a number: "))
7  print(factorial(n))

```

Output:



The screenshot shows a Windows command prompt window with the following content:

```
PS C:\Users\akshi\OneDrive\Desktop\full stack> & C:\Users\akshi\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/akshi/OneDrive/Desktop/ass - 7.4.py"
Enter a number: 5
120
PS C:\Users\akshi\OneDrive\Desktop\full stack>
```

The prompt shows the user is in the directory `C:\Users\akshi\OneDrive\Desktop\full stack`. They run a command to execute a Python script `c:/Users/akshi/OneDrive/Desktop/ass - 7.4.py`. The script prompts for a number, and the user enters `5`. The script then outputs `120`.

Explanation:

- The wrong code does not contain a base condition, so the recursive function keeps calling itself endlessly.
- This results in a RecursionError and program crash.
- The corrected code introduces a proper base case ($n == 0$ or $n == 1$).
- This base condition stops recursion at the correct point.
- As a result, the function now produces accurate factorial values.

Task 2: Fixing Data Type Errors in a Sorting Utility

Wrong Code:

```
ass - 7.4.py ●
C: > Users > akshi > OneDrive > Desktop > ass - 7.4.py > ...
1  def sort_list(data):
2      return sorted(data)
3
4  data = input("Enter values: ").split()
5  print(sort_list(data))
```

Output:

```
PS C:\Users\akshi\OneDrive\Desktop\full_stack> & C:/Users/akshi/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/akshi/OneDrive/Desktop/ass - 7.4.py"
Enter values: "2" "3" "2" "1"
['1', '2', '2', '3']
PS C:\Users\akshi\OneDrive\Desktop\full_stack>
```

Corrected Code:

```
ass - 7.4.py ✕
C: > Users > akshi > OneDrive > Desktop > ass - 7.4.py > ...
1  def sort_list(data):
2      cleaned = []
3      for i in data:
4          if i.isdigit():
5              cleaned.append(int(i))
6      return sorted(cleaned)
7
8  data = input("Enter values: ").split()
9  print(sort_list(data))
```

Output:

```
PS C:\Users\akshi\OneDrive\Desktop\full stack> & C:/Users/akshi/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/akshi/OneDrive/Desktop/ass - 7.4.py"
Enter values: 5 2 8 3 4 7
[2, 3, 4, 5, 7, 8]
PS C:\Users\akshi\OneDrive\Desktop\full stack> |
```

Explanation:

- The wrong code directly applies sorted() on mixed data types.
- Python cannot compare integers and strings, causing a TypeError
- The corrected code filters and converts valid numeric strings into integers.
- This ensures all elements are of the same data type before sorting.
- Thus, sorting is performed safely without runtime errors.

Task 3: Improving File Handling Reliability

Wrong Code:

```
ass - 7.4.py X
C: > Users > akshi > OneDrive > Desktop > ass - 7.4.py > ...
1 filename = input("Enter file name: ")
2 file = open(filename, "r")
3 print(file.read())
```

Output:

```
PS C:\Users\akshi\OneDrive\Desktop\full stack> & C:/Users/akshi/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/akshi/OneDrive/Desktop/ass - 7.4.py"
Enter file name: demo.py
Traceback (most recent call last):
  File "c:/Users/akshi/OneDrive/Desktop/ass - 7.4.py", line 2, in <module>
    file = open(filename, "r")
FileNotFoundError: [Errno 2] No such file or directory: 'demo.py'
PS C:\Users\akshi\OneDrive\Desktop\full stack> |
```

Corrected Code:

```
ass - 7.4.py X
C:\Users\akshi> OneDrive\Desktop> ass - 7.4.py > ...
1 filename = input("Enter file name: ")
2 with open(filename, "r") as file:
3     print(file.read())
```

Output:

```
PS C:\Users\akshi\OneDrive\Desktop\full stack> & C:/Users/akshi/Desktop/ass - 7.4.py
Enter file name: demo.js
function add(a, b) { return a + b; }
function sub(a, b) { return a + b; }
console.log(add(7, 5));
PS C:\Users\akshi\OneDrive\Desktop\full stack>
```

Explanation:

- The wrong code opens a file but does not explicitly close it.
- If an error occurs, the file remains open, causing resource leakage.
- The corrected code uses the `with open()` context manager.
- This automatically closes the file after execution.
- Hence, the program becomes safer and more reliable.

Task 4: Handling Runtime Errors Gracefully in Loops

Wrong Code:

```
ass - 7.4.py X
C: > Users > akshi > OneDrive > Desktop > ass - 7.4.py > ...
1  values = list(map(int, input("Enter numbers: ").split()))
2  for v in values:
3      print(10 / v)
```

Output:

```
PS C:\Users\akshi\OneDrive\Desktop\full stack> & C:/Users/akshi/AppData/Local
Desktop/ass - 7.4.py"
Enter numbers: 5 0 2 4
2.0
Traceback (most recent call last):
  File "c:\Users\akshi\OneDrive\Desktop\ass - 7.4.py", line 3, in <module>
    print(10 / v)
           ~~~~
ZeroDivisionError: division by zero
PS C:\Users\akshi\OneDrive\Desktop\full stack> 
```

Corrected Code:

```
ass - 7.4.py X
C: > Users > akshi > OneDrive > Desktop > ass - 7.4.py > [v] values
1  values = list(map(int, input("Enter numbers: ").split()))
2  for v in values:
3      try:
4          print(10 / v)
5      except ZeroDivisionError:
6          print("Cannot divide by zero")
```

Output:

```
PS C:\Users\akshi\OneDrive\Desktop\full stack> & C:/Use
Desktop/ass - 7.4.py"
Enter numbers: 5 0 2 4
2.0
Cannot divide by zero
5.0
2.5
PS C:\Users\akshi\OneDrive\Desktop\full stack> █
```

Explanation:

- The wrong code performs division without handling exceptions.
- When a zero value is encountered, the program crashes.
- The corrected code wraps the risky operation in a try-except block.
- This catches the ZeroDivisionError and displays a message instead.
- The loop continues execution for remaining values.

Task 5: Debugging Class Initialization Errors

Wrong Code:

```
ass - 7.4.py X
C: > Users > akshi > OneDrive > Desktop > ass - 7.4.py > ...
1  class Student:
2      def __init__(name, age):
3          name = name
4          age = age
5
6  name = input("Enter name: ")
7  age = int(input("Enter age: "))
8  s = Student(name, age)
9  print(s.name, s.age)
```

Output:

```
PS C:\Users\akshi\OneDrive\Desktop\full stack> & C:/Users/akshi/AppData/Local/Desktop/ass - 7.4.py"
Enter name: Sakshi
Enter age: 20
Traceback (most recent call last):
  File "c:\Users\akshi\OneDrive\Desktop\ass - 7.4.py", line 8, in <module>
    s = Student(name, age)
TypeError: Student.__init__() takes 2 positional arguments but 3 were given
PS C:\Users\akshi\OneDrive\Desktop\full stack> █
```

Corrected Code:

```
ass - 7.4.py X
C: > Users > akshi > OneDrive > Desktop > ass - 7.4.py > ...
1  class Student:
2      def __init__(self, name, age):
3          self.name = name
4          self.age = age
5
6  name = input("Enter name: ")
7  age = int(input("Enter age: "))
8  s = Student(name, age)
9  print(s.name, s.age)
```

Output:

```
PS C:\Users\akshi\OneDrive\Desktop\full stack> & C:/U
Desktop/ass - 7.4.py"
Enter name: Sakshi
Enter age: 20
Sakshi 20
PS C:\Users\akshi\OneDrive\Desktop\full stack> █
```

Explanation:

- The wrong code does not include the `self` parameter in the constructor.
- Attributes are not stored in the object, leading to access errors.
- The corrected code correctly uses `self` to define instance variables.
- This allows proper initialization of object attributes.
- As a result, object creation and attribute access work correctly.