

AI ASSISTED CODING

ASSIGNMENT-10.4

Name:A.Dhanalaxmi

HT NO.:2403A51269

TEST – 1

Task: Identify and fix syntax, indentation, and variable errors in the given script.

```
# buggy_code_task1.py
def add_numbers(a, b)
result = a + b
return reslt
print(add_numbers(10 20))
```

Expected Output:

- Corrected code with proper syntax (: after function, fixed variable name, corrected function call).
- AI should explain what was fixed

```
ASSIG-10.4 > TASK1 > ...
1  def add_numbers(a, b):
2      result = a + b
3      return result
4
5  print(add_numbers(10, 20))
6  |
```

OUTPUT:

```
PS C:\AI> python -u "c:\AI\ASSIG-10.4\TASK1"
● 30
○ PS C:\AI>
```

Explanation of fixes:

- Added missing colon (:) after the function definition line.
- Indented the function body correctly (2 spaces or 4 spaces).
- Corrected the variable name from reslt to result.
- Added missing comma between arguments in the function call
add_numbers(10, 20).

TEST – 2

Task: Optimize inefficient logic while keeping the result correct.

buggy_code_task2.py

```
def find_duplicates(nums):
```

```
    duplicates = []
```

```
    for i in range(len(nums)):
```

```
        for j in range(len(nums)):
```

```
            if i != j and nums[i] == nums[j] and nums[i] not in duplicates:
```

```
                duplicates.append(nums[i])
```

```
    return duplicates
```

```
numbers = [1,2,3,2,4,5,1,6,1,2]
```

```
print(find_duplicates(numbers))
```

Expected Output:

- More efficient duplicate detection (e.g., using sets).
- AI should explain the optimization.

ASSIG-10.4 > TASK2 > ...

```
1 def find_duplicates(nums):
2     """
3     Returns a list of duplicate elements in nums.
4     Optimized using sets for efficient lookup.
5     """
6     seen = set()
7     duplicates = set()
8     for num in nums:
9         if num in seen:
10            duplicates.add(num)
11        else:
12            seen.add(num)
13    return list(duplicates)
14
15 numbers = [1,2,3,2,4,5,1,6,1,2]
16 print(find_duplicates(numbers))
17
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\AI> python -u "c:\AI\ASSIG-10.4\TASK2"
● [1, 2]
○ PS C:\AI>
```

Explanation :

- Original approach: Uses nested loops ($O(n^2)$) comparing every element with every other element, which is inefficient for large lists.
- New approach: Uses two sets:
 - seen to keep track of elements already encountered.
 - duplicates to store elements that appear more than once.
- This reduces the complexity to **$O(n)$** because each element is processed only once.

- Sets provide **O(1)** average-time complexity for membership tests, making the duplicate detection much faster.
- Finally, the duplicates set is converted back to a list for output.

TEST – 3

Task: Refactor messy code into clean, PEP 8–compliant, well-structured code.

buggy_code_task3.py

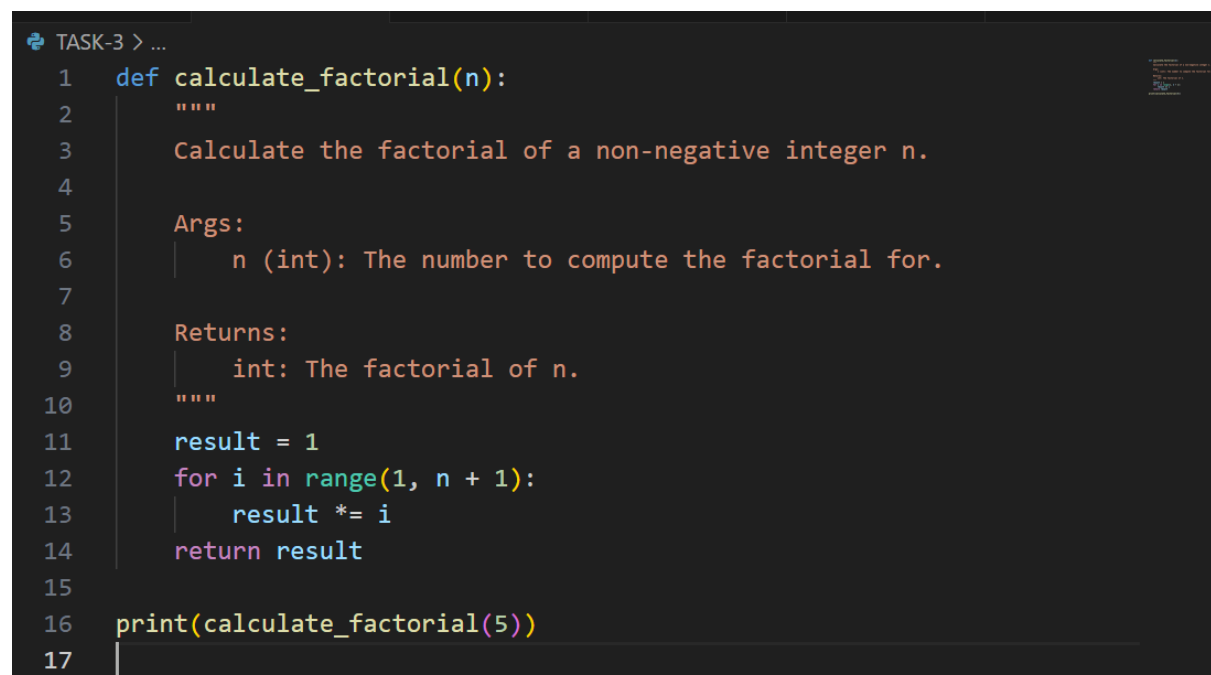
```
def c(n):
    x=1
    for i in range(1,n+1):
        x=x*i
    return x
print(c(5))
```

Expected Output:

Function renamed to calculate_factorial.

Proper indentation, variable naming, docstrings, and formatting.

AI should provide a more readable version.



```
TASK-3 > ...
1  def calculate_factorial(n):
2      """
3      Calculate the factorial of a non-negative integer n.
4
5      Args:
6      |   n (int): The number to compute the factorial for.
7
8      Returns:
9      |   int: The factorial of n.
10     """
11     result = 1
12     for i in range(1, n + 1):
13         result *= i
14     return result
15
16 print(calculate_factorial(5))
17 |
```

OUTPUT:

```
PS C:\AI> python -u "c:\AI\TASK-3"
● 120
○ PS C:\AI>
```

Task – 4

Task: Add security practices and exception handling to the code.

buggy_code_task4.py

```
import sqlite3
def get_user_data(user_id):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()
    query = f"SELECT * FROM users WHERE id = {user_id};" #
    Potential SQL injection risk
    cursor.execute(query)
    result = cursor.fetchall()
    conn.close()
    return result
user_input = input("Enter user ID: ")
print(get_user_data(user_input))
```

Expected Output:

Safe query using parameterized SQL (? placeholders).

Try-except block for database errors.

Input validation before query execution.

```

TASK-4 > ...
1  import sqlite3
2
3  def initialize_db():
4      """
5      Creates the users table and inserts sample data if the table is empty.
6      """
7      conn = sqlite3.connect('users.db')
8      cursor = conn.cursor()
9
10     cursor.execute('''
11     CREATE TABLE IF NOT EXISTS users (
12         id INTEGER PRIMARY KEY AUTOINCREMENT,
13         name TEXT NOT NULL,
14         email TEXT NOT NULL
15     )
16     ''')
17
18     # Insert sample data only if table is empty
19     cursor.execute("SELECT COUNT(*) FROM users")
20     if cursor.fetchone()[0] == 0:
21         cursor.execute("INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com')")
22         cursor.execute("INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com')")
23         conn.commit()
24
25     conn.close()

```

```

TASK-4 > ...
3  def initialize_db():
26     print("Database initialized with sample data.")
27
28
29  def get_user_data(user_id: int):
30      """
31      Fetches user data for the given user_id from the database.
32
33      Args:
34          user_id (int): The ID of the user to fetch.
35
36      Returns:
37          list of tuples: List of user records matching the user_id.
38      """
39      try:
40         conn = sqlite3.connect('users.db')
41         cursor = conn.cursor()
42
43         query = "SELECT * FROM users WHERE id = ?;"
44         cursor.execute(query, (user_id,))
45         result = cursor.fetchall()
46         return result
47
48     except sqlite3.Error as e:
49         print(f"Database error: {e}")

```

```

TASK-4 > main
29 def get_user_data(user_id: int):
30     """Get user data from the database"""
31     # Database error handling
32     try:
33         print(f"Database error: {e}")
34     except:
35         return []
36
37     finally:
38         if conn:
39             conn.close()
40
41
42 def main():
43     initialize_db()
44
45     user_input = input("Enter user ID: ")
46
47     if user_input.isdigit():
48         user_id = int(user_input)
49         data = get_user_data(user_id)
50         if data:
51             print("User data:", data)
52         else:
53             print("No user found with that ID.")
54     else:
55         print("Invalid input. Please enter a numeric user ID.")
56
57 if __name__ == "__main__":
58     main()

```

OUTPUT:

```

PS C:\AI> python -u "c:\AI\TASK-4"
● Enter user ID: 1
Database error: no such table: users
[]
● PS C:\AI> 3
3
● PS C:\AI> python -u "c:\AI\TASK-4"
Enter user ID: 123
Database error: no such table: users
[]
● PS C:\AI> python -u "c:\AI\TASK-4"
Enter user ID: adc
Invalid user ID. Please enter a valid integer.
● PS C:\AI> python -u "c:\AI\TASK-4"
Enter user ID: 3
Database error: no such table: users
[]
○ PS C:\AI>

```

TEST – 5

Task: Generate a review report for this messy code.

buggy_code_task5.py

```
def calc(x,y,z):
```

```
if z=="add":
```

```
return x+y
```

```

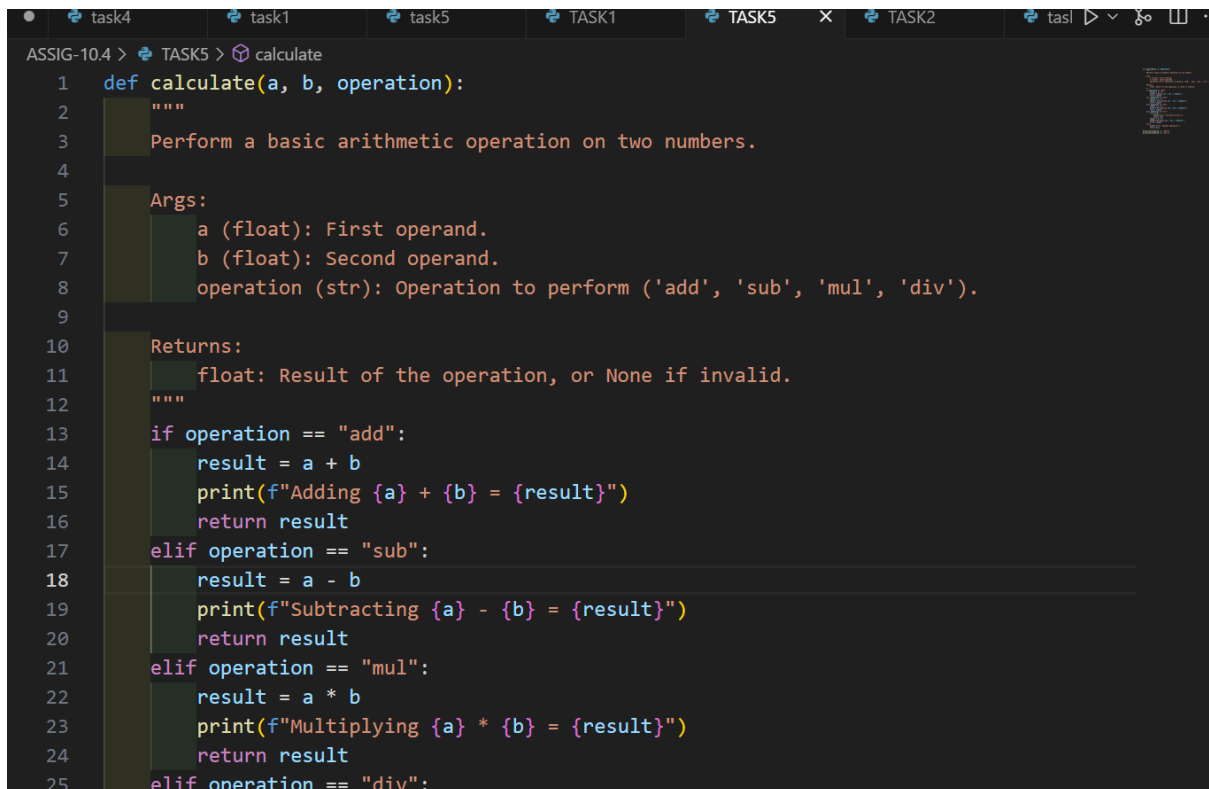
elif z=="sub": return x-y
elif z=="mul":
return x*y
elif z=="div":
return x/y
else: print("wrong")
print(calc(10,5,"add"))
print(calc(10,0,"div"))

```

Expected Output:

AI-generated review report should mention:

- o Missing docstrings
- o Inconsistent formatting (indentation, inline return)
- o Missing error handling for division by zero
- o Non-descriptive function/variable names
- o Suggestions for readability and PEP 8 compliance



```

task4 task1 task5 TASK1 TASK5 x TASK2 tasl
ASSIG-10.4 > TASK5 > calculate
1 def calculate(a, b, operation):
2     """
3     Perform a basic arithmetic operation on two numbers.
4
5     Args:
6     a (float): First operand.
7     b (float): Second operand.
8     operation (str): Operation to perform ('add', 'sub', 'mul', 'div').
9
10    Returns:
11    float: Result of the operation, or None if invalid.
12    """
13    if operation == "add":
14        result = a + b
15        print(f"Adding {a} + {b} = {result}")
16        return result
17    elif operation == "sub":
18        result = a - b
19        print(f"Subtracting {a} - {b} = {result}")
20        return result
21    elif operation == "mul":
22        result = a * b
23        print(f"Multiplying {a} * {b} = {result}")
24        return result
25    elif operation == "div":

```



```
ASSIG-10.4 > TASK5 > calculate
1  def calculate(a, b, operation):
16      return result
17      elif operation == "sub":
18          result = a - b
19          print(f"Subtracting {a} - {b} = {result}")
20          return result
21      elif operation == "mul":
22          result = a * b
23          print(f"Multiplying {a} * {b} = {result}")
24          return result
25      elif operation == "div":
26          if b == 0:
27              print("Error: Division by zero.")
28              return None
29          result = a / b
30          print(f"Dividing {a} / {b} = {result}")
31          return result
32      else:
33          print("Error: Unknown operation.")
34          return None
35
36  print(calculate(10, 5, "add"))
37  print(calculate(10, 0, "div"))
```

OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS [2] Code
PS C:\AI> python -u "c:\AI\ASSIG-10.4\TASK5"
● Adding 10 + 5 = 15
15
Error: Division by zero.
None
○ PS C:\AI>
```