

Name:Swetha Dixit

Course:AI Coding

Hall-Ticket:2303A52415

Batch:33

Assignment:10.2

Task:1

Task Description -1(Error Detection and Correction)

Task:

Use AI to analyze a Python script and correct all syntax and logical errors.

Sample Input Code:

```
def calculate_total(nums)
    sum = 0
    for n in nums
        sum += n
    return total
```

Expected Output-1:

Corrected and executable Python code with

brief explanations of the identified syntax and logic errors.

Python Code:

```
import unittest

def calculate_total(nums):
    total = 0
    for n in nums:
        total += n
    return total

class TestCalculateTotal(unittest.TestCase):

    def test_standard_list(self):
        # Test with positive integers
        self.assertEqual(calculate_total([1, 2, 3, 4, 5]), 15)

    def test_empty_list(self):
        # Test with an empty list (should return 0)
        self.assertEqual(calculate_total([]), 0)
```

```

def test_negative_numbers(self):

    # Test with negative integers

    self.assertEqual(calculate_total([-1, -2, -3]), -6)

def test_mixed_numbers(self):

    # Test with mixed positive and negative integers

    self.assertEqual(calculate_total([10, -5, 2]), 7)

# Running the tests

if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)

```

The screenshot shows the AI coding interface with the following components:

- Code Editor:** Displays the Python script `Task_1_10.2.py` containing the provided code for testing the `calculate_total` function.
- Terminal:** Shows the command-line output of running the test script. It includes environment variable information, the command run, and the output "Ran 4 tests in 0.000s".
- Dashboard:** Features a "Test cases for is_even function" section with a terminal input field, a "Thoughts" section, and a "Run your script from the terminal with:" section containing a command. It also displays a list of recent files and a "Agents" section.
- Bottom Bar:** Includes tabs for Problems, Output, Debug Console, Terminal, and Ports. It also shows the current file path: `C:\Users\intec\AI coding\Assignment 4.2> Task_1_10.2.py`.

The screenshot shows a code editor interface with several tabs at the top: k_5_72.py, D1_lab1.py, D2_lab1.py, Task_1_8.2.py, Task_2_8.2.py, Task_1_10.2.py, and Task_1_10.2.py (active). The main pane displays Python code:

```

Assignment 4.2 > Task_1_10.2.py > ...
1 def calculate_total(nums):
2     total = 0
3     for n in nums:
4         total += n
5     return total
6
7 # Example usage:
8 numbers = [1, 2, 3, 4, 5]
9 print(calculate_total(numbers)) # Output: 15

```

To the right of the code, there is a sidebar titled "Test cases for is_even function". It contains a message: "what I need to run in the terminal" followed by "python <path>". Below that, it says "Thought Is" and "Run your script from the terminal with:". A command "cd <path> & python Task_1_10.2.py" is shown. Further down, it says "If you're already in the Assignment 4.2 folder:" followed by "Right now the script only defines calculate_total and doesn't print anything. To see output, add a call at the end of the file (for example print(calculate_total([1, 2, 3])))".

At the bottom of the editor, there is a terminal window showing command-line interactions:

```

Problems Output Debug Console Terminal Ports
+ v ... x
powershell... powershell... powershell...
OK
(.venv) PS C:\Users\intec\AI coding\Assignment 4.2> & "C:\Users\intec\AI coding\.venv\Scripts\python.exe" "c:/Users/intec/AI coding/Assignment 4.2/Task_1_10.2.py"
15
Ran 4 tests in 0.000s
OK
(.venv) PS C:\Users\intec\AI coding\Assignment 4.2> & "C:\Users\intec\AI coding\.venv\Scripts\python.exe" "c:/Users/intec/AI coding/Assignment 4.2/Task_1_10.2.py"
15
OK
(.venv) PS C:\Users\intec\AI coding\Assignment 4.2>

```

Analysis of Errors

Here is a breakdown of the specific syntax and logical errors identified and corrected:

1. Syntax Errors (Prevent the code from running)

- **Missing Colons:** The original code lacked colons (:) at the end of the function definition (`def ...`) and the loop statement (`for ...`). In Python, control flow statements must end with a colon.
 - *Fix:* Added : to lines 1 and 3.

2. Logical Errors (Code runs but fails or gives wrong results)

- **Undefined Variable in Return Statement:** The function attempted to `return total`, but the variable used to accumulate the sum was named `sum`. This would cause a `NameError` because `total` did not exist within the function's scope.
 - *Fix:* Updated the return statement to match the accumulator variable.
- **Shadowing Built-in Functions (Best Practice):** The original code used `sum` as a variable name. While valid, this "shadows" Python's built-in `sum()` function, which is bad practice and can lead to bugs later.
 - *Fix:* Renamed the variable from `sum` to `total`.

Task:2 Task Description -2(Code Style Standardization)

Task:

Use AI to refactor Python code to comply with standard coding style guidelines.

Sample Input Code:

```
def findSum(a,b):return a+b  
print(findSum(5,10))
```

Expected Output-2:

Well-structured, consistently formatted Python code following standard style conventions.

Python code:

```
def calculate_sum(a: int, b: int) -> int:  
    """  
    Calculates the sum of two numbers.  
  
    Args:  
        a (int): The first number.  
        b (int): The second number.  
  
    Returns:  
        int: The sum of a and b.  
    """  
  
    return a + b
```

```

if __name__ == "__main__":
    result = calculate_sum(5, 10)
    print(result)

```

The screenshot shows the AI coding interface. On the left, the code editor displays a Python script named `Task_2_10.2.py` with the following content:

```

1 def calculate_sum(a: int, b: int) -> int:
2     """
3         Calculates the sum of two numbers.
4
5     Args:
6         a (int): The first number.
7         b (int): The second number.
8
9     Returns:
10        int: The sum of a and b.
11        """
12
13     return a + b
14
15 if __name__ == "__main__":
16     result = calculate_sum(5, 10)
17     print(result)

```

The right side of the interface includes a sidebar with "Test cases for is_even function" and a terminal window showing command-line history:

```

OK
● (.venv) PS C:\Users\intec\AI coding\Assignment 4.2> & "C:\Users\intec\AI coding\.venv\scripts\python.exe" "c:/Users/intec/AI coding/Assignment 4.2/task_1_10.2.py"
15
● (.venv) PS C:\Users\intec\AI coding\Assignment 4.2> & "C:\Users\intec\AI coding\.venv\scripts\python.exe" "c:/Users/intec/AI coding/Assignment 4.2/task_2_10.2.py"
15
● (.venv) PS C:\Users\intec\AI coding\Assignment 4.2> & "C:\Users\intec\AI coding\.venv\scripts\python.exe" "c:/Users/intec/AI coding/Assignment 4.2/task_2_10.2.py"
15
○ (.venv) PS C:\Users\intec\AI coding\Assignment 4.2>

```

Key Style Improvements Applied (PEP 8)

- **Naming Convention (Snake Case):** Changed `findSum` (camelCase) to `calculate_sum` (snake_case). In Python, functions and variables should be lowercase with underscores.
- **Whitespace and Indentation:**
 - Moved the function body (`return a + b`) to a new line with a 4-space indentation.
 - Added spaces around operators (`+`) and after commas (e.g., `a, b`) to improve readability.

- **Documentation (Docstrings):** Added a docstring ("'''...'''") to explain the function's purpose, arguments, and return value.
- **Type Hinting:** Added `: int` and `-> int` annotations. While not strictly mandatory in PEP 8, they are highly recommended in modern Python for clarity and debugging.
- **Execution Block:** Wrapped the executable code in `if __name__ == "__main__":`. This prevents the code from running automatically if this script is imported as a module into another program.

Task:3

Task Description -3(Code Clarity Improvement)

Task:

Use AI to improve code readability without changing its functionality.

Sample Input Code:

```
def f(x,y):  
  
    return x-y*2  
  
print(f(10,3))
```

Expected Output-3:

Python code rewritten with meaningful function and variable names, proper indentation, and improved clarity.

```
def subtract_doubled_value(initial_number, value_to_double):  
    """  
        Subtracts double the second argument from the first argument.  
  
        Formula: initial_number - (value_to_double * 2)  
    """  
  
    doubled_amount = value_to_double * 2  
  
    result = initial_number - doubled_amount  
  
    return result  
  
  
# Example usage:  
  
# Calculation: 10 - (3 * 2) = 4  
  
print(subtract_doubled_value(10, 3))
```

```

1 def subtract_doubled_value(initial_number, value_to_double):
2     """
3         Subtracts double the second argument from the first argument.
4     """
5     Formula: initial_number - (value_to_double * 2)
6     """
7     doubled_amount = value_to_double * 2
8     result = initial_number - doubled_amount
9     return result
10
11 # Example usage:
12 # calculation: 10 - (3 * 2) == 4
13 print(subtract_doubled_value(10, 3))

```

Test cases for is_even function

what I need to run in the terminal

Thought Is

Run your script from the terminal with:

```
python "c:\Users\intec\AI coding\Assignment 4.2\Task_1_10.2.py"
```

If you're already in the Assignment 4.2 folder:

```
cd "c:\Users\intec\AI coding\Assignment 4.2"
python Task_1_10.2.py
```

Right now the script only defines calculate_total and doesn't print anything. To see output, add a call at the end of the file (for example `print(calculate_total([1, 2, 3]))`).

Problems Output Debug Console Terminal Ports

```

.exe" "c:/Users/intec/AI coding/Assignment 4.2/Task_1_10.2.py"
15
● (.venv) PS C:\Users\intec\AI coding\Assignment 4.2 & "c:/Users/intec/AI coding/.venv\Scripts\python
.exe" "c:/Users/intec/AI coding/Assignment 4.2/Task_2_10.2.py"
15
● (.venv) PS C:\Users\intec\AI coding\Assignment 4.2 & "c:/Users/intec/AI coding/.venv\Scripts\python
.exe" "c:/Users/intec/AI coding/Assignment 4.2/Task_2_10.2.py"
15
● (.venv) PS C:\Users\intec\AI coding\Assignment 4.2 & "c:/Users/intec/AI coding/.venv\Scripts\python
● .exe" "c:/Users/intec/AI coding/Assignment 4.2/task_3_10.2.py"
4
○ (.venv) PS C:\Users\intec\AI coding\Assignment 4.2> []

```

powerhell... ▲
powerhell... □
powershell

> 4 Files Undo All Keep All Review

Plan, ⚙ for context, / for commands

Agent Auto

Cursor Tab 1n 13, Col 37 (402 selected) Spaces:4 UTF-8 ↻ Python .venv (3.14.0) ⌂ Go Live

Key Improvements Made

- **Meaningful Naming:**

- Changed the function name `f` to `subtract_doubled_value` so the user knows exactly what the function does without reading the logic.
- Changed `x` and `y` to `initial_number` and `value_to_double`. This prevents confusion about which number is being subtracted and which is being doubled.

- **Intermediate Variables:**

- Introduced `doubled_amount` and `result`. Breaking the calculation into steps makes the logic easier to follow during debugging.

- **Spacing and Indentation:**

- Added spaces around operators (`*`, `-`) for visual breathing room.
- Corrected the indentation of the `return` statement (standard Python indentation is 4 spaces).

- **Documentation:**
 - Added a docstring to explain the formula being used.

Task:4 Task Description -4(Structural Refactoring)

Task:

Use AI to refactor repetitive code into reusable functions.

Sample Input Code:

```
print("Hello Ram")
```

```
print("Hello Sita")
```

```
print("Hello Ravi")
```

Expected Output-4:

Modular Python code using reusable functions to eliminate repetition

Python code:

```
class Greeter:  
  
    def __init__(self, greeting_style="Hello"):  
  
        """  
  
        Initialize the Greeter with a specific style.  
  
        Default is 'Hello'.  
  
        """  
  
        self.greeting_style = greeting_style
```

```
def greet(self, name):  
    """  
    Prints the greeting using the stored style.  
    """  
  
    print(f"{self.greeting_style} {name}")  
  
# Usage 1: Default Greeter  
  
default_greeter = Greeter()  
  
users = ["Ram", "Sita", "Ravi"]  
  
print("--- Default Greetings ---")  
  
for user in users:  
  
    default_greeter.greet(user)  
  
# Usage 2: Custom Greeter (demonstrating reusability)  
  
formal_greeter = Greeter(greeting_style="Welcome")  
  
print("\n--- Formal Greetings ---")  
  
for user in users:  
  
    formal_greeter.greet(user)
```

```
1  class Greeter:
2      def __init__(self, greeting_style="Hello"):
3          """
4              Initialize the Greeter with a specific style.
5              Default is 'Hello'.
6          """
7          self.greeting_style = greeting_style
8
9      def greet(self, name):
10         """
11             Prints the greeting using the stored style.
12         """
13         print(f"{self.greeting_style} {name}")
14
15 # Usage 1: Default Greeter
16 default_greeter = Greeter()
17 users = ["Ram", "Sita", "Ravi"]
18
19 print("--- Default Greetings ---")
20 for user in users:
21     default_greeter.greet(user)
22
23 # Usage 2: Custom Greeter (demonstrating reusability)
24 formal_greeter = Greeter(greeting_style="Welcome, ")
25
```

Problems Output Debug Console Terminal Ports

(.venv) PS C:\Users\intec\Ai coding\Assignment 4.2> & "C:\Users\intec\Ai coding\.venv\Scripts\python.exe" "c:/Users/intec/Ai coding/Assignment 4.2/Task_4_10.2.py"

--- Default Greetings ---

Hello Ram
Hello Sita
Hello Ravi

--- Formal Greetings ---

Welcome, Ram
Welcome, Sita
Welcome, Ravi

(.venv) PS C:\Users\intec\Ai coding\Assignment 4.2>

State Retention: The `Greeter` remembers the `greeting_style`. You don't have to pass "Hello" or "Welcome" every time you call the function.

Extensibility: You could easily add methods like `change_style()` or a counter to track how many people were greeted without changing the external code structure.

Task:5 Task Description -5(Efficiency Enhancement)

Task:

Use AI to optimize Python code for better performance.

Sample Input Code:

```
numbers = [ ]  
  
for i in range(1, 500000):  
  
    numbers.append(i * i)  
  
print(len(numbers))
```

Expected Output-5:

Optimized Python code that achieves the same result with improved performance.

Python code:

```
# Using a Generator Expression  
  
# Note: sum(1 for ...) is a common idiom to count generator items  
  
count = sum(1 for _ in range(1, 500000))  
  
print(count)
```

A screenshot of a code editor interface, likely VS Code, showing a Python file named `Task_5_10.2.py`. The code uses a generator expression to sum 1 for each item in a range from 1 to 500,000. The terminal below shows the output of running the script, which prints "Hello Ram", "Hello Sita", "Hello Ravi", and a formal greeting message. It also shows the command used to run the script and the resulting output of 499999.

```
Assignment 4.2 > Task_5_10.2.py > ...
1 # Using a Generator Expression
2 # Note: sum(1 for ...) is a common idiom to count generator items
3 count = sum(1 for _ in range(1, 500000))
4 print(count)

Problems Output Debug Console Terminal Ports + ... ^ x
Hello Ram
Hello Sita
Hello Ravi
--- Formal Greetings ---
Welcome, Ram
Welcome, Sita
Welcome, Ravi
● (.venv) PS C:\Users\intec\Ai coding\Assignment 4.2> & "C:\Users\intec\Ai coding\.venv\Scripts\python.exe" "c:/Users/intec/Ai coding/Assignment 4.2/Task_5_10.2.py"
499999
○ (.venv) PS C:\Users\intec\Ai coding\Assignment 4.2> []
Ctrl+K to generate command
>< Ai coding × 0 △ 0
```

Original Code: The `.append()` method is called 499,999 times. Each call involves a small amount of overhead that adds up.

List Comprehension: Pre-allocates memory chunks and avoids the repeated attribute lookup for `.append`.