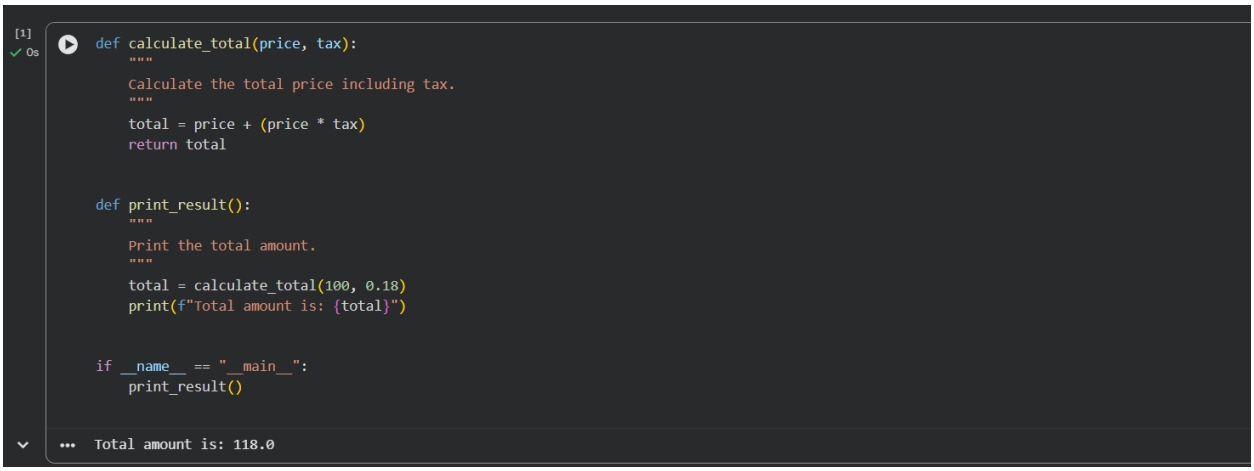
ASSIGNMENT 10.4

Task 1: AI-Assisted Syntax and Code Quality Review
Scenario
You join a development team and are asked to review a junior developer's Python script that fails to run correctly due to basic coding mistakes. Before deployment, the code must be corrected and Standardized

CODE/OUTPUT:



```
[1]
0s
def calculate_total(price, tax):
    """
    Calculate the total price including tax.
    """
    total = price + (price * tax)
    return total


def print_result():
    """
    Print the total amount.
    """
    total = calculate_total(100, 0.18)
    print(f"Total amount is: {total}")


if __name__ == "__main__":
    print_result()

Total amount is: 118.0
```

Explanation:

- The function `calculate_total` calculates the total price by adding tax to the original price.
- The function `print_result` calls `calculate_total` with a price of 100 and a tax rate of 18%.
- It then prints the final total amount.
- The program runs `print_result` to display the result.

Task 2: Performance-Oriented Code Review

Scenario

A data processing function works correctly but is inefficient and slows down the system when large datasets are used.

CODE/OUTPUT:

```
[2]
0s
def find_duplicates_slow(data):
    """
    Inefficient method using nested loops (O(n^2)).
    Returns a list of duplicate values.
    """
    duplicates = []

    for i in range(len(data)):
        for j in range(i + 1, len(data)):
            if data[i] == data[j] and data[i] not in duplicates:
                duplicates.append(data[i])

    return duplicates


def find_duplicates_fast(data):
    """
    Optimized method using sets (O(n)).
    Returns a list of duplicate values.
    """
    seen = set()
    duplicates = set()

    for item in data:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)
```



```
0s
            seen.add(item)

    return list(duplicates)


def main():
    data = [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1, 9]

    print("Original data:", data)
    print("Duplicates (slow method):", find_duplicates_slow(data))
    print("Duplicates (fast method):", find_duplicates_fast(data))


if __name__ == "__main__":
    main()

...
Original data: [1, 2, 3, 4, 2, 5, 6, 3, 7, 8, 1, 9]
Duplicates (slow method): [1, 2, 3]
Duplicates (fast method): [1, 2, 3]
```

Explanation:

- The program finds duplicate elements in a list using two different methods.
- `find_duplicates_slow` uses nested loops to compare each element with others, which is slower for large datasets.
- `find_duplicates_fast` uses sets to track seen values, making it much faster and more efficient.

  The `main` function demonstrates both methods and prints the duplicate values found in the list.

Task 3: Readability and Maintainability Refactoring

Scenario
A working script exists in a project, but it is difficult to understand due to poor naming, formatting, and structure. The team wants it rewritten for long-term maintainability

CODE/OUTPUT:

```
[3]
0s
def f(a, b):
    x = 0
    for i in a:
        if i > b:
            x = x + i
    return x


def calculate_sum_greater_than_threshold(numbers, threshold):
    total_sum = 0
    for number in numbers:
        if number > threshold:
            total_sum += number
    return total_sum


def main():
    data = [10, 5, 20, 3, 15, 7]
    threshold_value = 10
    print(calculate_sum_greater_than_threshold(data, threshold_value))


if __name__ == "__main__":
    main()

35
```

EXPLANATION:

- The program calculates the sum of numbers in a list that are greater than a given threshold value.
- The function `calculate_sum_greater_than_threshold` iterates through the list and adds only the values that exceed the threshold.
- The `main` function defines the input list and threshold, then calls the function and prints the result.
- This refactored version uses clear naming and structured logic, making the code easy to read and maintain.

Task 4: Secure Coding and Reliability Review

Scenario

A backend function retrieves user data from a database but has security vulnerabilities and poor error handling, making it unsafe for production deployment

CODE /OUTPUT:

```python
import sqlite3
def get_user_insecure(user_id):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()

    query = f"SELECT * FROM users WHERE id = {user_id}"
    cursor.execute(query)

    result = cursor.fetchone()
    conn.close()
    return result

def get_user_secure(user_id):
    if not isinstance(user_id, int):
        raise ValueError("Invalid user ID. Must be an integer.")

    try:
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()

        query = "SELECT * FROM users WHERE id = ?"
        cursor.execute(query, (user_id,))

        result = cursor.fetchone()
        return result
```

```python
    except sqlite3.Error as error:
        print("Database error:", error)
        return None

    finally:
        if conn:
            conn.close()

def main():
    user_id = 1
    print("Secure user data:", get_user_secure(user_id))

if __name__ == "__main__":
    main()
```

```
Database error: no such table: users
Secure user data: None
```

Explanation:

- The insecure function builds SQL queries using string formatting, which can lead to SQL injection attacks.
- The secure function uses parameterized queries to safely pass user input to the database.
- Input validation ensures that the user ID is of the correct type before executing the query.
- Try-except blocks handle database errors safely, making the code more reliable and suitable for production.

Task 5: AI-Based Automated Code Review Report

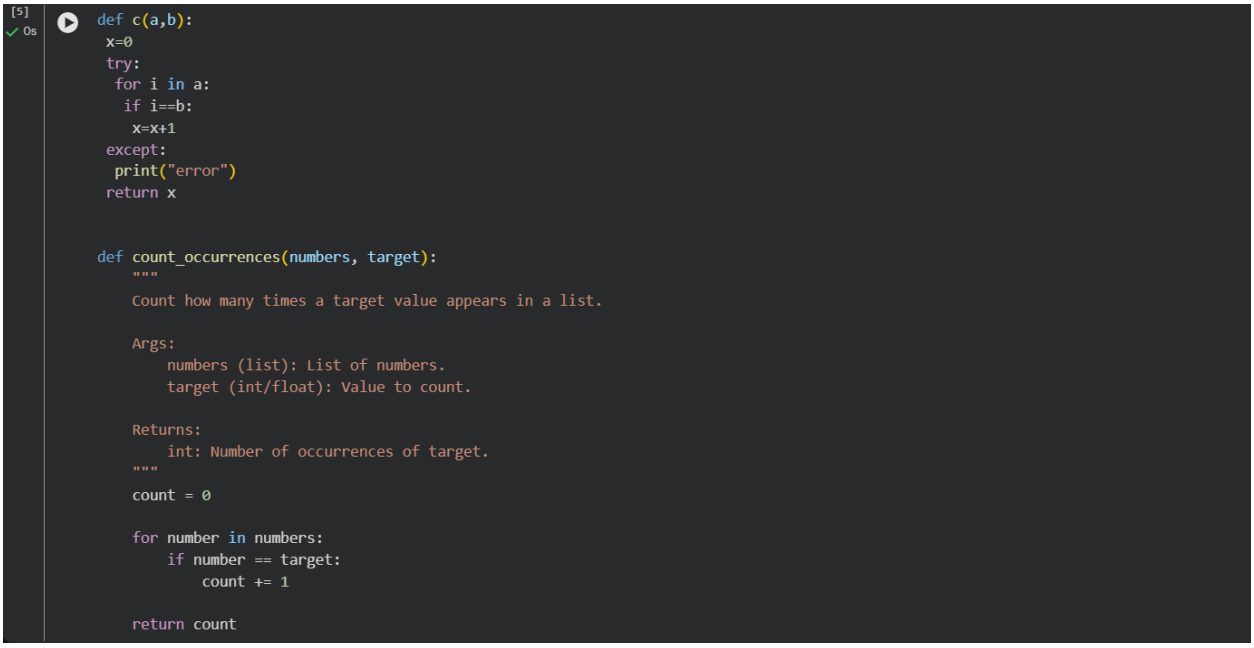
Scenario

Your team uses AI tools to perform automated preliminary code reviews

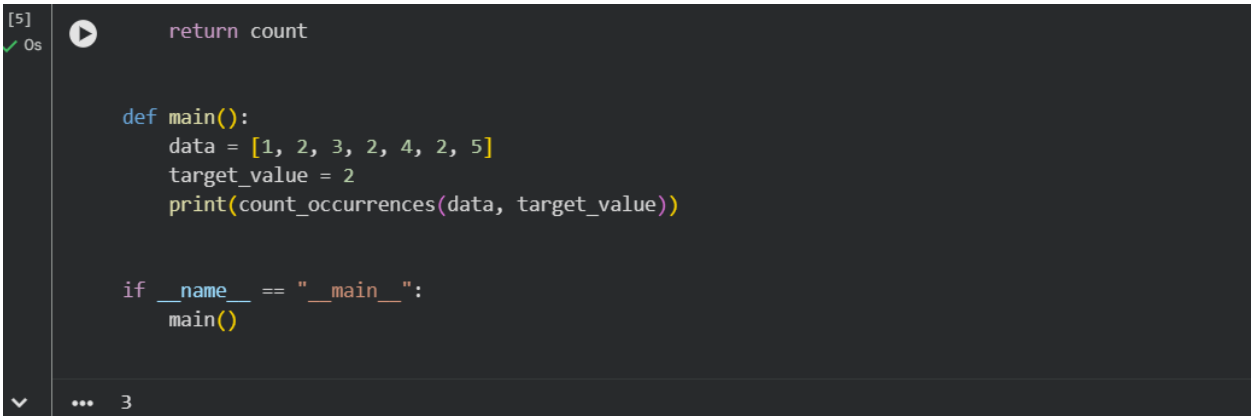
before human review, to improve code quality and consistency across

Projects.

CODE/OUTPUT:



```
[5]
✓ 0s
def c(a,b):
 x=0
 try:
  for i in a:
   if i==b:
    x=x+1
 except:
  print("error")
 return x


def count_occurrences(numbers, target):
    """
    Count how many times a target value appears in a list.

    Args:
        numbers (list): List of numbers.
        target (int/float): Value to count.

    Returns:
        int: Number of occurrences of target.
    """
    count = 0

    for number in numbers:
        if number == target:
            count += 1

    return count
```



Explanation:

- The original code had poor naming, bad formatting, and unclear logic, making it hard to understand and maintain.
- The improved version uses meaningful function and variable names, proper indentation, and clear structure.
- A docstring was added to explain the purpose and usage of the function.
- Overall, the refactored code is more readable, maintainable, and consistent with Python coding standards.