ASSIGNMENT 10.4

D.Charmi

2303A51314

Batch – 05

TASK – 01:

Wrong Code :

```python
def addNumbers(a b)
result = a + b
return result

def main():
num1 = 10
num2 = "20"
sum = addnumbers(num1, num2)
print("Sum is", sum)

main()
```

Output :

```
PROBLEMS 4  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Coding/10.4.py"
  File "c:\Users\devis\OneDrive\Desktop\AI Assisted Coding\10.4.py", line 1
    def addNumbers(a b)
                     ^
SyntaxError: invalid syntax
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Corrected Code:

```python
#Correct the above code to properly define the function and call it with correct parameters and types.
def add_numbers(a, b):
    result = a + b
    return result
def main():
    num1 = 10
    num2 = 20
    sum_result = add_numbers(num1, num2)
    print("Sum is", sum_result)
main()
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Coding/10.4.py"
Sum is 30
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

The given program has syntax errors like missing commas and wrong indentation.
The function name is written differently while calling it, causing an error.
A string value is added to an integer, which leads to a type error.
Overall, the code cannot run because of these basic mistakes.

All syntax and indentation errors are fixed.
Function and variable names are corrected and standardized.
Both values are integers, so addition works correctly.
The program now runs successfully and prints the correct sum.

TASK – 02:

Inefficient Code:

```
10.4.py > ...
def find_duplicates(data):
    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
# Example usage:
data = [1, 2, 3, 4, 2, 5, 1]
duplicates = find_duplicates(data)
print("Duplicate elements:", duplicates)

```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/devis/OneDrive/Desktop/AI Assisted Cod
ing/10.4.py"
Duplicate elements: [1, 2]
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

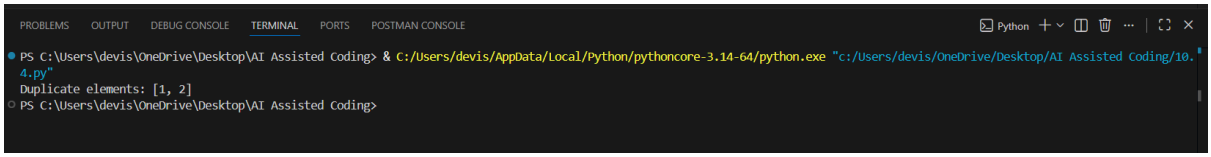
Improved version of Code :

```

#Improve the function by using a set to track seen elements and duplicates, which will reduce the time complexity to O(n):
def find_duplicates(data):
    seen = set()
    duplicates = set()
    for item in data:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)
    return list(duplicates)
# Example usage:
data = [1, 2, 3, 4, 2, 5, 1]
duplicates = find_duplicates(data)
print("Duplicate elements:", duplicates)
'''
    This function takes a list of data as input and returns a list of duplicate elements found in the input list.

    :param data: List of elements to check for duplicates
    :return: A list of duplicate elements
    '''
```

Output :



Explanation :

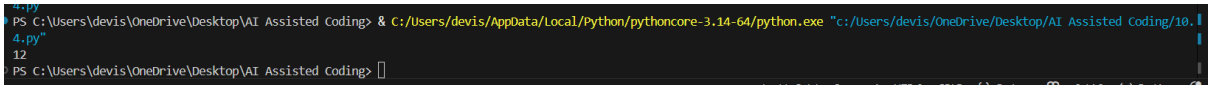
- The original function uses nested loops, which compare each element with every other element, resulting in O(n^2) time complexity.
- This approach becomes slow and inefficient when the input list is large.
- The improved function uses sets to track seen elements and duplicates.
- Set lookups are fast (O(1)), so each element is checked only once.
- This reduces the overall time complexity to O(n).
- The optimized function produces the same correct output but with much better performance and readability.

TASK – 03:

Poorly Structured Code :

```
10.4.py > ...
1   def f(a):
2    x=0
3    for i in a:
4     if i%2==0:
5      x=x+i
6    return x
7
8   # Example usage:
9   a=[1,2,3,4,5,6]
10  print(f(a))
11
```

Output :



Refactored Code :

```python
#Refactor the above  code with proper naming conventions
def sum_of_even_numbers(numbers):
    '''
    This function takes a list of numbers as input and returns the sum of even numbers.

    :param numbers: List of integers
    :return: Sum of even numbers
    '''
    even_sum = 0
    for number in numbers:
        if number % 2 == 0:
            even_sum += number
    return even_sum
# Example usage:
numbers = [1, 2, 3, 4, 5, 6]
print(sum_of_even_numbers(numbers))

```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/d
4.py"
12
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

- The unclear function name f was renamed to sum_of_even_numbers to clearly describe its purpose.
- Variable names like a, x, and i were replaced with meaningful names (numbers, even_sum, number).
- Proper indentation and formatting were applied for better readability.
- A docstring was added to explain the function's purpose, parameters, and return value.
- The logic remains the same, but the code is now easy to read and maintain.

TASK – 04

Insecure Code :

```
10.4.py > ...
import sqlite3
conn = sqlite3.connect("shop.db")
cursor = conn.cursor()
cursor.execute("""
CREATE TABLE IF NOT EXISTS products (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    price REAL
)
""")
cursor.execute("INSERT INTO products (name, price) VALUES (?, ?)",
               ("Laptop", 75000))
cursor.execute("INSERT INTO products (name, price) VALUES (?, ?)",
               ("Phone", 35000))
conn.commit()
conn.close()
print("Products table created successfully!")
def get_product(product_name):
    conn = sqlite3.connect("shop.db")
    cursor = conn.cursor()
    query = "SELECT * FROM products WHERE name = '" + product_name + "'"
    cursor.execute(query)
    result = cursor.fetchall()
    conn.close()
    return result
# Example usage
name = input("Enter product name: ")
print(get_product(name))
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/A
4.py"
Products table created successfully!
Enter product name: Laptop
[(1, 'Laptop', 75000.0), (3, 'Laptop', 75000.0)]
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Secured Code:

```python
#Secure version using parameterized queries
def get_product_secure(product_name):
    conn = sqlite3.connect("shop.db")
    cursor = conn.cursor()
    query = "SELECT * FROM products WHERE name = ?"
    cursor.execute(query, (product_name,))
    result = cursor.fetchall()
    conn.close()
    return result
# Example usage
name = input("Enter product name: ")
print(get_product_secure(name))
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devis/A
4.py"
Products table created successfully!
Enter product name: Laptop
[(1, 'Laptop', 75000.0), (3, 'Laptop', 75000.0)]
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

- The code contains two different versions of the same functionality.
- The first version is an insecure example.
- It uses string concatenation to build SQL queries, which is unsafe and vulnerable to SQL injection.
- The second version is the secure implementation that actually runs.
- It uses parameterized queries (?), which safely handle user input.
- The secure version is safe, reliable, and suitable for production use.

TASK – 05:

Poor Code:

```
10.4.py > ...
1   def p(a,b):
2    r=0
3    for i in a:
4     if i==b:
5      r=r+1
6    return r
7   '''    Docstring for p
8
9       This function counts the occurrences of a specific element in a list.
10
11      :param a: List of elements
12      :param b: Element to count
13      :return: Count of occurrences of b in a
14      '''
15  # Example usage:
16  a=[1,2,3,4,1,2,1]
17  b=1
18  count=p(a,b)
19  print("Count of", b, "in list:", count)
20  print(p.__doc__)
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Use
4.py"
Count of 1 in list: 3
None
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Improved version :

```
21
22  #Improve the above code in terms of readability and documentation.
23  def count_occurrences(elements, target):
24      '''
25      This function counts the occurrences of a specific element in a list.
26
27      :param elements: List of elements
28      :param target: Element to count
29      :return: Count of occurrences of target in elements
30      '''
31      count = 0
32      for element in elements:
33          if element == target:
34              count += 1
35      return count
36  # Example usage:
37  elements = [1, 2, 3, 4, 1, 2, 1]
38  target = 1
39  occurrences = count_occurrences(elements, target)
40  print("Count of", target, "in list:", occurrences)
41  print(count_occurrences.__doc__)
42
```

Output :

```
PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding> & C:/Users/devi
4.py"
Count of 1 in list: 3

This function counts the occurrences of a specific element in a list.

:param elements: List of elements
:param target: Element to count
:return: Count of occurrences of target in elements

PS C:\Users\devis\OneDrive\Desktop\AI Assisted Coding>
```

Explanation :

- The unclear function name p was renamed to count_occurrences to describe its purpose clearly.
- Variable names a and b were replaced with meaningful names (elements, target).
- Proper indentation and formatting were applied for better readability.
- A clear and correct docstring was added inside the function.
- The code logic remains the same, but it is now easy to understand and maintain.