ASSIGNMENT-7.5

2303A51635

BATCH NO:23

S.KEERTHI

Task 1: (Mutable Default Argument – Function Bug)

 Task: Analyze given code where a mutable default argument causes unexpected behavior. Use AI to fix it.

\# Bug: Mutable default argument

def add_item(item, items=[]):

  items.append(item)

  return items

print(add_item(1))

print(add_item(2))

 Expected Output: Corrected function avoids shared list bug.

```
#TASK1
#don't use mutable default arguments
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
```

OUTPUT:

```
PS C:\Users\sande\OneDrive\Desktop\AI ASSI>
 .py"
 [1]
 [1, 2]
```
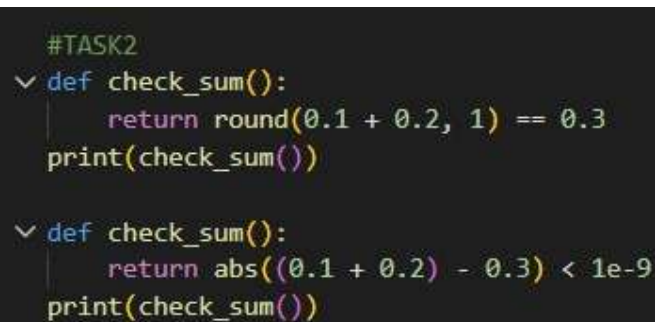
Task 2 :(Floating-Point Precision Error)

Task: Analyze given code where floating-point comparison fails.   Use AI to correct with tolerance.
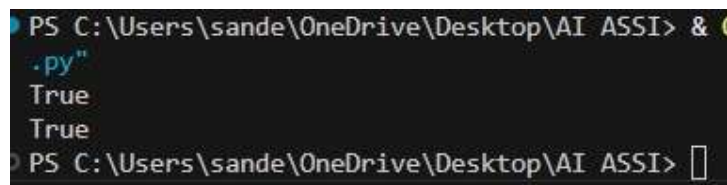
Bug: Floating point precision issue

def check_sum():

    return (0.1 + 0.2) == 0.3

print(check_sum())

Expected Output: Corrected function

```
#TASK2
v def check_sum():
      return round(0.1 + 0.2, 1) == 0.3
  print(check_sum())

v def check_sum():
      return abs((0.1 + 0.2) - 0.3) < 1e-9
  print(check_sum())
```

OUTPUT:

```
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> & (
.py"
True
True
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> []
```

Task 3: (Recursion Error – Missing Base Case)

 Task: Analyze given code where recursion runs infinitely due to missing base case. Use AI to fix.

 # Bug: No base case

def countdown(n):

  print(n)

  return countdown(n-1)

countdown(5)

Expected Output : Correct recursion with stopping condition.

```
#TASK3
def countdown(n):
    if n == 0:
        print(0)
        return
    print(n)
    countdown(n - 1)
countdown(5)
```

OUTPUT:

```
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
.py"
5
4
3
2
1
0
```

Task 4 (Dictionary Key Error)

Task: Analyze given code where a missing dictionary key causes error. Use AI to fix it.

Bug: Accessing non-existing key

def get_value():

   data = {"a": 1, "b": 2}

   return data["c"]

print(get_value())

Expected Output: Corrected with .get() or error handling.

```
#TASK4
def get_value():
    data = {"a": 1, "b": 2}
    return data.get("c", "Key not found")
print(get_value())
```

OUTPUT:

```
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
 .py"
Key not found
```

Task 5: (Infinite Loop – Wrong Condition)

Task: Analyze given code where loop never ends. Use AI to detect and fix it.

 # Bug: Infinite loop

def loop_example():

  i = 0

  while i < 5:

    print(i)

Expected Output: Corrected loop increments i.

```
#TASK5
#fix the bug in the code
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1
loop_example()
```

OUTPUT:

```
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> & C:/Users
 .py"
0
1
2
3
4
```

Task 6 (Unpacking Error – Wrong Variables)

Task: Analyze given code where tuple unpacking fails. Use AI to fix it.

 # Bug: Wrong unpacking

a, b = (1, 2, 3)

Expected Output: Correct unpacking or using _ for extra values.

```
#TASK6
#fix the bug in the code
a, b, c = (1, 2, 3)
print(a, b, c)
```

OUTPUT:

```
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> & C
 .py"
 1 2 3
```

Task 7 (Mixed Indentation – Tabs vs Spaces)

Task: Analyze given code where mixed indentation breaks execution. Use AI to fix it.

 # Bug: Mixed indentation

def func():

  x = 5

    y = 10

  return x+y

Expected Output : Consistent indentation applied.

```
#TASK7
#fix the indentation error in the code
def func():
    x = 5
    y = 10
    return x+y
result = func()
print(result)
```

OUTPUT:
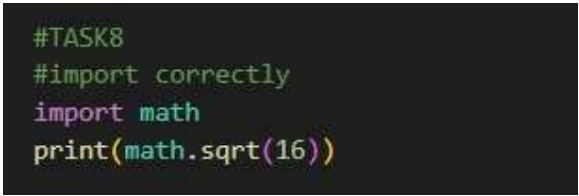
```
15
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
 .py"
 15
```

Task 8 (Import Error – Wrong Module Usage)

Task: Analyze given code with incorrect import. Use AI to fix.
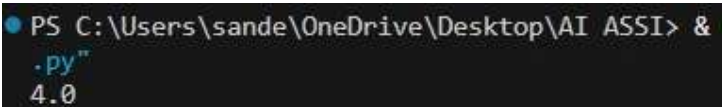
 # Bug: Wrong import

import maths

print(maths.sqrt(16))

Expected Output: Corrected to import math

```
#TASK8
#import correctly
import math
print(math.sqrt(16))
```

OUTPUT:

```
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
.py"
4.0
```

Task 9 (Unreachable Code – Return Inside Loop)

Task: Analyze given code where a return inside a loop prevents full iteration. Use AI to fix it.

 # Bug: Early return inside loop

def total(numbers):

  for n in numbers:

    return n

print(total([1,2,3]))

Expected Output: Corrected code accumulates sum and returns after loop.

```
#TASK9
# #fix the bug in the code
∨ def total(numbers):
      total = 0
∨     for n in numbers:
          total += n
      return total
  print(total([1,2,3]))
```

OUTPUT:

```
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
 .py"
 6
```

Task 10 (Name Error – Undefined Variable)

Task: Analyze given code where a variable is used before being defined. Let AI detect and fix the error.

Bug: Using undefined variable

def calculate_area():

   return length * width

print(calculate_area())

Requirements:

- Run the code to observe the error.
- Ask AI to identify the missing variable definition.
- Fix the bug by defining length and width as parameters.
- Add 3 assert test cases for correctness.

Expected Output :

- Corrected code with parameters.
- AI explanation of the bug.

Successful execution of assertions.

```
#TASK10
#fix the bug in the code
from turtle import width
def calculate_area(length, width):
    return length * width
print(calculate_area(5, 3))
```

OUTPUT:

```
15
● PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
  .py"
  15
```

Task 11 (Type Error – Mixing Data Types Incorrectly)

Task: Analyze given code where integers and strings are added incorrectly. Let AI detect and fix the error.

Bug: Adding integer and string

def add_values():

   return 5 + "10"

print(add_values())

Requirements:

- Run the code to observe the error.
- AI should explain why int + str is invalid.
- Fix the code by type conversion (e.g., int("10") or str(5)).
- Verify with 3 assert cases.

Expected Output #6:

- Corrected code with type handling.
- AI explanation of the fix.

Successful test validation.

```
#TASK11
#fix the bug in the code
def add_values():
    return 5 + 10
print(add_values())
```

OUTPUT:

```
15
● PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
  .py"
  15
```

Task 12 (Type Error – String + List Concatenation)

Task: Analyze code where a string is incorrectly added to a list.

 # Bug: Adding string and list

def combine():

   return "Numbers: " + [1, 2, 3]

print(combine())

Requirements:

- **Run the code to observe the error.**
- **Explain why str + list is invalid.**
- **Fix using conversion (str([1,2,3]) or " ".join()).**
- **Verify with 3 assert cases.**

Expected Output:

- **Corrected code**
- **Explanation**

Successful test validation

```
#TASK12
#fix the bug in the code
def combine():
    numbers = [1, 2, 3]
    return f"Numbers: {numbers}"
print(combine())
```

OUTPUT:

```
● PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
  .py"
  Numbers: [1, 2, 3]
```

Task 13 (Type Error – Multiplying String by Float)

Task: Detect and fix code where a string is multiplied by a float.

 # Bug: Multiplying string by float

def repeat_text():

  return "Hello" * 2.5

print(repeat_text())

Requirements:

- **Observe the error.**
- **Explain why float multiplication is invalid for strings.**
- **Fix by converting float to int.**

Add 3 assert test cases

```
#TASK13
#fix the bug in the code
def repeat_text():
    return "Hello" * 2
print(repeat_text())
```

OUTPUT:

```
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
.py"
HelloHello
```

Task 14 (Type Error – Adding None to Integer)

Task: Analyze code where None is added to an integer.

 # Bug: Adding None and integer

def compute():

  value = None

  return value + 10

print(compute())

Requirements:

- **Run and identify the error.**
- **Explain why NoneType cannot be added.**
- **Fix by assigning a default value.**
- **Validate using asserts.**

```
#TASK14
#fix the bug in the code
def compute():
    value = 5
    return value + 10
print(compute())
```

OUTPUT:

```
15
● PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
  .py"
  15
```

Task 15 (Type Error – Input Treated as String Instead of Number)

Task: Fix code where user input is not converted properly.

 # Bug: Input remains string

def sum_two_numbers():

  a = input("Enter first number: ")

  b = input("Enter second number: ")

  return a + b

print(sum_two_numbers())

Requirements:

- **Explain why input is always string.**
- **Fix using int() conversion.**

Verify with assert test cases

```
#TASK15
def sum_two_numbers():
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    return a + b
print(sum_two_numbers())
```

OUTPUT:

```
PS C:\Users\sande\OneDrive\Desktop\AI ASSI> &
.py"
Enter first number: 8
Enter second number: 9
17
```