AI -ASSISTED-CODING

ASSIGNMENT-7.1

2403A51246

B PRANEETH

Batch-11

Task-1 :

Task Description #1 (Syntax Errors – Missing Parentheses in Print Statement)

Task: Provide a Python snippet with a missing parenthesis in a print statement (e.g., print "Hello"). Use AI to detect and fix the syntax error.

```
# Bug: Missing parentheses in print statement
def greet():
    print "Hello, AI Debugging Lab!"
greet()
```

Requirements:

- Run the given code to observe the error.
- Apply AI suggestions to correct the syntax.
- Use at least 3 assert test cases to confirm the corrected code works.

Expected Output #1:

- Corrected code with proper syntax and AI explanation.

PROMPT :

#Provide a Python snippet with a missing parenthesis in a print statement

#e.g. print "Hello,AI Debugging Lab!"

#Bug: Missing parenthesis at the end of the print statement

CODE:

```
ai.py > ...
#Provide a Python snippet with a missing parenthesis in a print statement
#e.g. print "Hello,AI Debugging Lab!"
#Missing parenthesis at the end of the print statement
def greet():
    print "Hello, AI Debugging Lab!"
greet()
```

Explanation:

ERROR :

→Missing paranthesis in print statement

- print "Hello,AI Debugging Lab!"

CORRECTION :

→Replaced the old-style print with Python 3 style print(...).

→Returned the greeting string from the function so we can test it using assert.

→Added assert statements to ensure the function behaves as expected

Corrected code:

1.

```python
#Provide a Python snippet with a missing parenthesis in a print statement
#e.g. print "Hello,AI Debugging Lab!"
#Bug: Missing parenthesis at the end of the print statement
def greet():
    print("Hello, AI Debugging Lab!")
greet()
```

2.

```python
#Provide a Python snippet with a missing parenthesis in a print statement
#e.g. print "Hello,AI Debugging Lab!
def greet():
    return "Hello, AI Debugging Lab!"
assert greet() == "Hello, AI Debugging Lab!"
assert isinstance(greet(), str)
assert "AI Debugging Lab" in greet()
print(greet())
```

OUTPUT:

Hello,AI Debugging Lab!

TASK-2:

Task: Supply a function where an if-condition mistakenly uses = instead of ==. Let AI identify and fix the issue.

Bug: Using assignment (=) instead of comparison (==)

def check_number(n):

if n = 10:

return "Ten"

else:

return "Not Ten"

Requirements:

• Ask AI to explain why this causes a bug.

• Correct the code and verify with 3 assert test cases.

Expected Output #2:

• Corrected code using == with explanation and successful test execution

PROMPT:

#Supply a function where an if-condition mistakenly uses = instead of ==. Let AI identify and fix the issue.

Bug: Using assignment (=) instead of comparison (==)

CODE:

```python
#Supply a function where an if-condition mistakenly uses = instead of ==. Let AI identify a
# Bug: Using assignment (=) instead of comparison (==)
def check_value(x):
    if x = 10:
        return "x is ten"
    else:
        return "x is not ten"
check_value(10)
```

Corrected code:

```python
#Supply a function where an if-condition mistakenly uses = instead of ==. Let AI identify a
# Bug: Using assignment (=) instead of comparison (==)
def check_value(x):
    if x == 10:
        return "x is ten"
    else:
        return "x is not ten"
check_value(10)
```

Explantion:

- In Python, the single equals sign = is used for assignment, not comparison.
- In an if statement, Python expects a boolean expression, like n == 10.

- Writing if n = 10: tries to assign 10 to n inside the if, which is not allowed and results in a syntax error.

OUTPUT:

10 is ten

TASK-3:

Provide code that attempts to open a non-existent file and crashes. Use AI to apply safe error handling.

```
# Bug: Program crashes if file is missing
def read_file(filename):
with open(filename, 'r') as f:
return f.read()
print(read_file("nonexistent.txt"))
```

Requirements:

• Implement a try-except block suggested by AI.

• Add a user-friendly error message.

• Test with at least 3 scenarios: file exists, file missing, invalid path.

Expected Output #3:

• Safe file handling with exception management.

PROMPT:

#write a Python function that attempts to read a file, but it crashes if the file does not exist

Bug: Program crashes if file is missing

Code:

```
ai.py > ...
#write a Python function that attempts to read a file, but it crashes if the file does not exist
# Bug: Program crashes if file is missing
def read_file(file_path):
    with open(file_path, 'r') as file:
        content = file.read()
    return content
print(read_file('non_existent_file.txt'))
```

Error :

FileNotFoundError: [Errno 2] No such file or directory: 'nonexistent.txt'

Correction in code:

```
ai.py > ...
#write a Python function that attempts to read a file, but it crashes if the file does not exist
# Bug: Program crashes if file is missing
# Fixed: Added error handling to manage missing file scenario
def read_file_fixed(file_path):

    try:
        with open(file_path, 'r') as file:
            content = file.read()
        return content
    except FileNotFoundError:
        return "Error: The file does not exist."

print(read_file_fixed('non_existent_file.txt'))
```

Explanation :

Using try-except to catch FileNotFoundError and other exceptions to prevent the program from crashing.

Expected behavior:

- File content prints if file exists.
- User-friendly error message prints if file missing or invalid path.
- No uncaught exceptions.

OUTPUT :

```
This is a test file.
Error: The file 'nonexistent.txt' was not found.
Error: The file '/invalid/path/to/file.txt' was not found.
```

TASK-4:

Give a class where a non-existent method is called (e.g., obj.undefined_method()). Use AI to debug and fix.

```
# Bug: Calling an undefined method
class Car:
def start(self):
return "Car started"
my_car = Car()
print(my_car.drive()) # drive() is not defined
```

Requirements:

• Students must analyze whether to define the missing method or correct the method call.

• Use 3 assert tests to confirm the corrected class works.

Expected Output #4:

• Corrected class with clear AI explanation.

PROMPT :

#write a Python class with a bug: a method is being called on an object, but that method **is not defined** in the class.

#Bug: Calling an undefined method

CODE :

```
+# Bug: Calling an undefined method
+class Car:
+  def start(self):
+    return "Car started"
+
+my_car = Car()
+# This will cause an AttributeError because drive() is not defined
+print(my_car.drive())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipython-input-2494917172.py in <cell line: 0>()
      6 my_car = Car()
      7 # This will cause an AttributeError because drive() is not defined
----> 8 print(my_car.drive())

AttributeError: 'Car' object has no attribute 'drive'
```

Next steps: Explain error

CORRECTION IN CODE AND OUTPUT :

```
+# Bug: Calling an undefined method
+class Car:
+  def start(self):
+    return "Car started"
+
+  # Option 2: Define the missing method (uncomment the following if needed)
+  # def drive(self):
+  #   return "Car is driving"
+
+
+my_car = Car()
+# This will cause an AttributeError because drive() is not defined
+# print(my_car.drive())
+
+# Corrected code (assuming you meant to call the start method)
+print(my_car.start())
+
+# Add assert test cases for the corrected code
+assert my_car.start() == "Car started", "Test Case 1 Failed"
+# Add more test cases if other methods were defined or expected
+
+print("All test cases passed!") # You can remove this line if you don't want the explicit success message
```

```
Car started
All test cases passed!
```

TASK-5 :

Provide code that adds an integer and string ("5" + 2) causing a TypeError. Use AI to resolve the bug.

Bug: TypeError due to mixing string and integer

def add_five(value):

return value + 5

print(add_five("10"))

Requirements:

• Ask AI for two solutions: type casting and string concatenation.

• Validate with 3 assert test cases.

Expected Output #5:

• Corrected code that runs successfully for multiple inputs.

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

Criteria Max Marks

Identification of bugs 0.5

Application of AI-suggested fixes 0.5

Explanation and understanding of

errors 0.5
Corrected code functionality 0.5
Report structure and reflection 0.5
Total 2.5 Marks

PROMPT:

#write a python for the task involving a TypeError from adding a string and an integer.

Bug: TypeError due to mixing string and integer

CODE :

```
+# Bug: TypeError due to mixing string and integer
+def add_five(value):
+  return value + 5
+
+# This will cause a TypeError
+print(add_five("10"))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-2288065219.py in <cell line: 0>()
      4
      5 # This will cause a TypeError
----> 6 print(add_five("10"))

/tmp/ipython-input-2288065219.py in add_five(value)
      1 # Bug: TypeError due to mixing string and integer
      2 def add_five(value):
----> 3   return value + 5
      4
      5 # This will cause a TypeError

TypeError: can only concatenate str (not "int") to str
```

Next steps: Explain error

CORRECTION IN CODE :

```
+# Bug: TypeError due to mixing string and integer
+
+# Solution 1: Type Casting
+# Explanation: Convert the string to an integer before adding.
+def add_five_type_casting(value):
+  try:
+    return int(value) + 5
+  except ValueError:
+    return "Error: Cannot convert input to an integer."
+
+# Solution 2: String Concatenation
+# Explanation: Convert the integer to a string before concatenating.
+def add_five_string_concatenation(value):
+  try:
+    return value + str(5)
+  except TypeError:
+     return "Error: Input must be a string for concatenation."
+
+
+# Test cases for Solution 1 (Type Casting)
+print("Testing Type Casting Solution:")
+print(add_five_type_casting("10"))
+print(add_five_type_casting("0"))
+print(add_five_type_casting("-5"))
+print(add_five_type_casting("abc")) # Test with invalid input
+
+# Test cases for Solution 2 (String Concatenation)
+print("\nTesting String Concatenation Solution:")
+print(add_five_string_concatenation("10"))
+print(add_five_string_concatenation("0"))
+print(add_five_string_concatenation("-5"))
+# print(add_five_string_concatenation(123)) # Uncomment to see TypeError if input is not a string
+
+# Add assert test cases (for type casting as it's more likely the intended math operation)
+assert add_five_type_casting("10") == 15, "Type Casting Test Case 1 Failed"
+assert add_five_type_casting("0") == 5, "Type Casting Test Case 2 Failed"
+assert add_five_type_casting("-5") == 0, "Type Casting Test Case 3 Failed"
+assert add_five_type_casting("abc") == "Error: Cannot convert input to an integer.", "Type Casting Test Case 4 Failed"
+
+print("\nAll type casting test cases passed!")
```

OUTPUT :

```
Testing Type Casting Solution:
15
5
0
Error: Cannot convert input to an integer.

Testing String Concatenation Solution:
105
05
-55

All type casting test cases passed!
```