



**Explanation:** In Python 3, `print` is defined as a built-in function. A function call requires parentheses. Using the Python 2 `print` statement format violates Python 3 syntax rules, resulting in a `SyntaxError`. The correction is to use the functional form of `print()`.

### Task Description #2 (Incorrect condition in an If Statement)

**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.

#### Output Screenshot:

The screenshot shows a code editor with the following content:

```
1 def check_number(n):  
2     if n == 10:  
3         return "Ten"  
4     else:  
5         return "Not Ten"  
6  
7 # Assert Test Cases  
8 assert check_number(10) == "Ten"  
9 assert check_number(5) == "Not Ten"  
10 assert check_number(-10) == "Not Ten"  
11  
12 print("Task 2 tests passed!")  
13
```

The terminal output at the bottom shows the execution of the code:

```
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding> & "C:\Users\Lenovo\Desktop\AI Coding\.venv\Scripts\python.exe" "c:\Users\Lenovo\Desktop\AI Coding\ass5.py\assignment 7.1.py\task1.py"  
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding> & "C:\Users\Lenovo\Desktop\AI Coding\.venv\Scripts\python.exe" "c:\Users\Lenovo\Desktop\AI Coding\ass5.py\assignment 7.1.py\task2.py"  
Task 2 tests passed!
```

#### Explanation:

The operator `=` is an assignment operator used to store a value in a variable. Conditional statements require a boolean expression, which is formed using comparison operators such as `==`. Using `=` in an if-condition is syntactically invalid in Python and produces a `SyntaxError`. The correction is to replace `=` with `==`.

### Task Description #3 (Runtime Error – File Not Found)

**Task:** 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.

### Output Screenshot:

```
1 def read_file(filename):
2     # 1) Handle invalid input like empty string or only spaces
3     if not isinstance(filename, str) or filename.strip() == "":
4         return "Error: Invalid file path."
5
6     # 2) Safe file handling
7     try:
8         with open(filename, "r") as f:
9             return f.read()
10
11     except FileNotFoundError:
12         return f"Error: File '{filename}' not found."
13
14     except OSError:
15         return f"Error: Invalid file path '{filename}'."
16
17 # Assert Test Cases
18 assert read_file("") == "Error: Invalid file path."
19 assert read_file(" ") == "Error: Invalid file path."
20 assert read_file("non_existent_file.txt") == "Error: File 'non_existent_file.txt' not found."
21 print("All tests passed!")
```

Terminal Output: vo/Desktop/AI Coding/ass5.py/assignment 7.1.py/task3.py  
All tests passed!  
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding>

**EXPLANATION :** File operations depend on the existence and validity of the file path. When `open()` is executed with a missing file, Python raises a `File Not Found Error` at runtime. Exception handling using `try-except` prevents abrupt termination and enables controlled execution by returning a meaningful error message.

### Task Description #4 (Calling a Non-Existent Method)

**Task:** 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.

## Output Screenshot:

```

1 class Car:
2     def start(self):
3         return "Car started"
4
5     def drive(self):
6         return "Car is driving"
7
8 # Object
9 my_car = Car()
10
11 # Output
12 print(my_car.start())
13
14 # Assert Test Cases
15 assert my_car.start() == "Car started"
16 assert my_car.drive() == "Car is driving"
17 assert isinstance(my_car.start(), str)
18
19 print("Task 4 tests passed!")
20

```

```

(.venv) PS C:\Users\Lenovo\Desktop\VAI Coding> & "C:\Users\Lenovo\Desktop\VAI Coding\ass5.py\assignment 7.1.py\task4.py"
Car is driving
Task 4 tests passed!
(.venv) PS C:\Users\Lenovo\Desktop\VAI Coding>

```

**Explanation:** In object-oriented programming, a method must be defined within a class before it can be invoked by an object of that class. Calling an undefined method results in an `AttributeError` because the object does not contain the requested attribute. The correction requires either defining the missing method in the class or modifying the call to an existing method.

## Task Description #5 (TypeError – Mixing Strings and Integers in Addition)

**Task:** 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.

## Output Screenshot:

```
File Edit Selection View Go Run ... AI Coding
task2.py ...assignment 7.1.py task3.py ...assignment 7.1.py task4.py ...assignment 7.1.py task5.py ...assignment 7.1.py
EXPLORER
OPEN EDITORS
AI CODING
.env
ass5.py
assignment 7.1.py
task1.py
task2.py
task3.py
task4.py
task5.py
.env
ass_5 task1.py
ass_5 task2.py
ass_5 task3.py
ass_5 task4.py
ass_5 task5.py
ass4.3 task1.py
ass4.3 task2.py
ass4.3 task3.py
ass4.3 task4(4.1).py
ass4.3 task4(4.2).py
ass4.3 task5.py
ass6.3 task1.py
ass6.3 task2.py
ass6.3 task3.py
ass6.3 task4.py
ass6.3 task5.py
OUTLINE
TIMELINE
ass5.py > assignment 7.1.py > task5.py > ...
1 def add_five_cast(value):
2     return int(value) + 5
3
4 # Assert Test Cases
5 assert add_five_cast("10") == 15
6 assert add_five_cast(0) == 5
7 assert add_five_cast("25") == 30
8
9 print("Task 5 (casting) tests passed!")
10
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding> & "C:\Users\Lenovo\Desktop\AI Coding\.venv\Scripts\python.exe" "c:/Users/Leno
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding> & "C:\Users\Lenovo\Desktop\AI Coding\.venv\Scripts\python.exe" "c:/Users/Leno
vo/Desktop/AI Coding/ass5.py/assignment 7.1.py/task5.py"
Task 5 (casting) tests passed!
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding>
```

**EXPLANATION :** Python enforces strict type rules for arithmetic operations. Addition between a string and an integer is not supported because the operands are of incompatible types. This produces a `TypeError`. The correction is performed by explicit type conversion, either converting the string to an integer for numeric addition or converting the integer to a string for concatenation.