

# ASSIGNMENT – 10.4

NAME: NALLALA SIRI

HT.NO: 2403A52037

BATCH: 03

## TASK 1:

Identify and fix syntax, indentation, and variable errors in the given script.

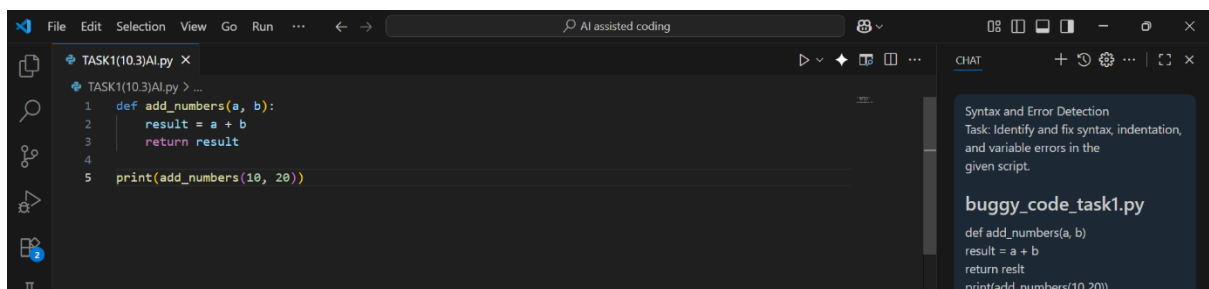
```
# buggy_code_task1.py
def add_numbers(a, b)
result = a + b
return reslt
print(add_numbers(10 20))
```

## PROMPT:

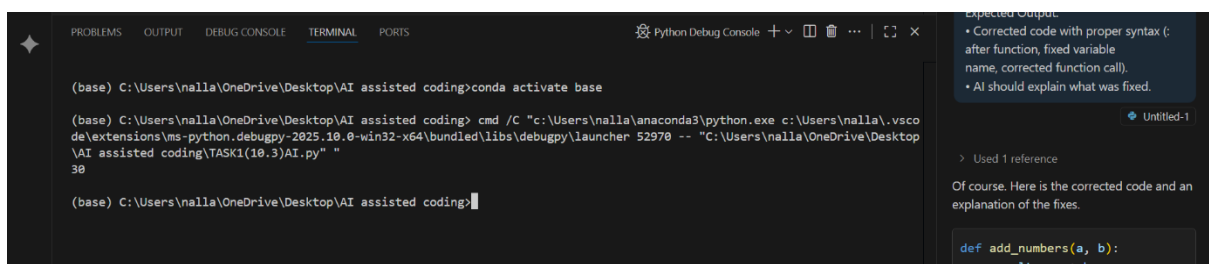
Detect and correct the syntax, indentation and variable errors from the given code:

```
def add_numbers(a, b)
result = a + b
return reslt
print(add_numbers(10 20))
```

## CODE:



## OUTPUT:



## OBSERVATION:

The colon is missed after the function definition and didn't indent the code block. It also had a typo in the return statement, using `reslt` instead of `result`. Finally, the comma between the arguments at the `add_numbers` function.

## TASK 2:

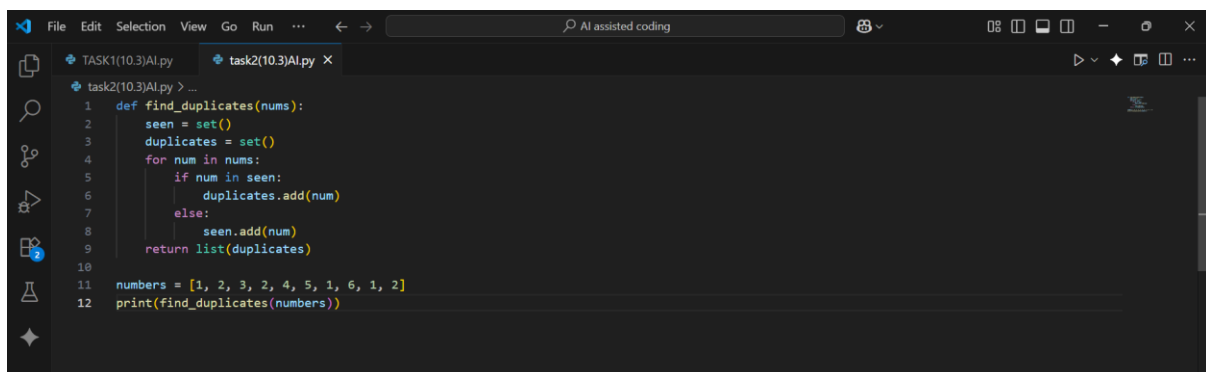
Optimize inefficient logic while keeping the result correct.

```
# buggy_code_task2.py
def find_duplicates(nums):
    duplicates = []
    for i in range(len(nums)):
        for j in range(len(nums)):
            if i != j and nums[i] == nums[j] and nums[i] not in duplicates:
                duplicates.append(nums[i])
    return duplicates
numbers = [1,2,3,2,4,5,1,6,1,2]
print(find_duplicates(numbers))
```

## PROMPT:

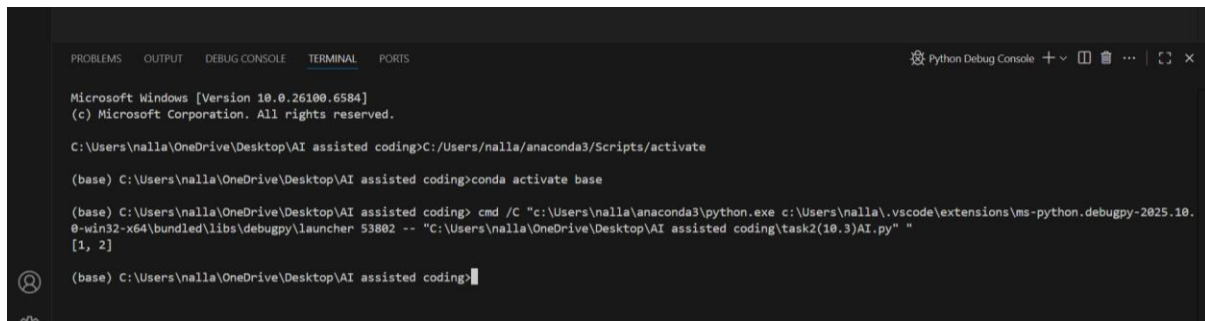
Write a Python function to find all duplicate elements in a list. The function should take a list of numbers as input and return a list containing only the numbers that appear more than once. Your solution should be efficient and avoid nested loops. For example, for the input `[1, 2, 3, 2, 4, 5, 1, 6, 1, 2]`, the function should return `[1, 2]`.

## CODE:

A screenshot of a code editor window with a dark theme. The editor shows a Python file named 'task2(10.3)AI.py'. The code defines a function 'find\_duplicates' that uses a set to track seen numbers and a list to collect duplicates. It then applies this function to a list of numbers and prints the result. The code is as follows:

```
1 def find_duplicates(nums):
2     seen = set()
3     duplicates = set()
4     for num in nums:
5         if num in seen:
6             duplicates.add(num)
7         else:
8             seen.add(num)
9     return list(duplicates)
10
11 numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
12 print(find_duplicates(numbers))
```

## OUTPUT:



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalla\OneDrive\Desktop\AI assisted coding>C:/Users/nalla/anaconda3/Scripts/activate

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher 53802 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\task2(10.3)AI.py" "
```

## OBSERVATION:

The Counter class from this module is specifically designed to count hashable objects, making it perfect for this task. By creating a Counter object from the input list, you get a dictionary-like mapping of each number to its frequency. A simple list comprehension can then iterate through this counter's items, returning a new list that includes only those numbers with a count greater than one. This approach is not only more compact, fitting into a single line, but is also highly readable and idiomatic in Python for frequency-related problems, achieving the same efficient  $O(n)$  time complexity as the set-based solution.

## TASK 3:

Refactor messy code into clean, PEP 8–compliant, well-structured code.

```
# buggy_code_task3.py
```

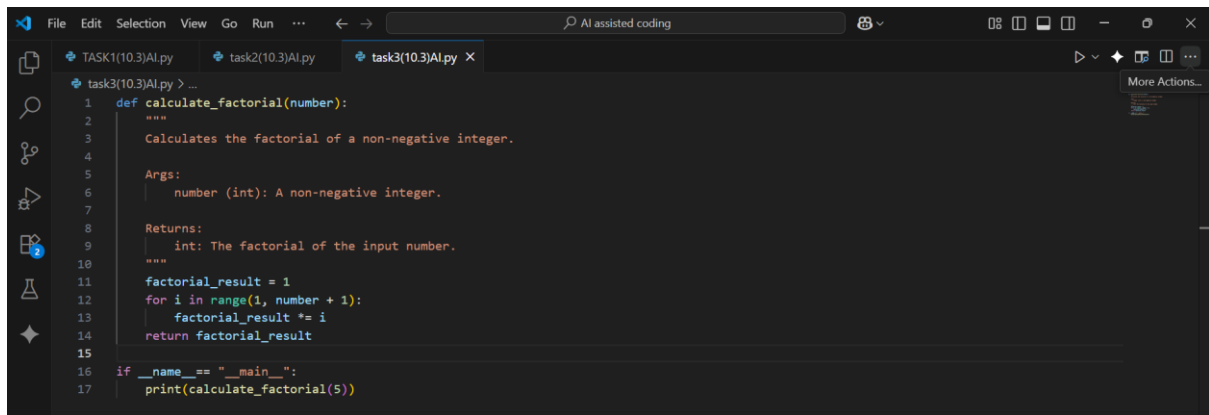
```
def c(n):
    x=1
    for i in range(1,n+1):
        x=x*i
    return x
print(c(5))
```

## PROMPT:

Refactor the following Python code to improve readability and adhere to PEP 8 standards. Rename the function and variables to be more descriptive, add a comprehensive docstring, and wrap the main execution block appropriately.

```
def c(n):
    x=1
    for i in range(1,n+1):
        x=x*i
    return x
print(c(5))
```

## CODE:



```
1 def calculate_factorial(number):
2     """
3     Calculates the factorial of a non-negative integer.
4
5     Args:
6         number (int): A non-negative integer.
7
8     Returns:
9         int: The factorial of the input number.
10    """
11    factorial_result = 1
12    for i in range(1, number + 1):
13        factorial_result *= i
14    return factorial_result
15
16 if __name__ == "__main__":
17     print(calculate_factorial(5))
```

## OUTPUT:



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalla\OneDrive\Desktop\AI assisted coding>C:\Users\nalla\anaconda3\Scripts\activate

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>cmd /C "C:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher 63595 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\task3(10.3)AI.py" "
120

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>
```

## OBSERVATION:

This Python script defines a function `calculate_factorial` that computes the factorial of a given non-negative integer by initializing a result to 1 and then iteratively multiplying it by each integer from 1 up to the input number. The script then uses a standard `if __name__ == "__main__":` block, which ensures the code within it only runs when the file is executed directly. Inside this block, it calls the `calculate_factorial` function with the number 5 and prints the resulting value, 120, to the console.

## TASK 4:

Add security practices and exception handling to the code.

```
# buggy_code_task4.py
```

```
import sqlite3
```

```
def get_user_data(user_id):
```

```
    conn = sqlite3.connect("users.db")
```

```
    cursor = conn.cursor()
```

```
    query = f"SELECT * FROM users WHERE id = {user_id};" #
```

Potential SQL injection risk

```
    cursor.execute(query)
```

```
    result = cursor.fetchall()
```

```
    conn.close()
```

```
    return result
```

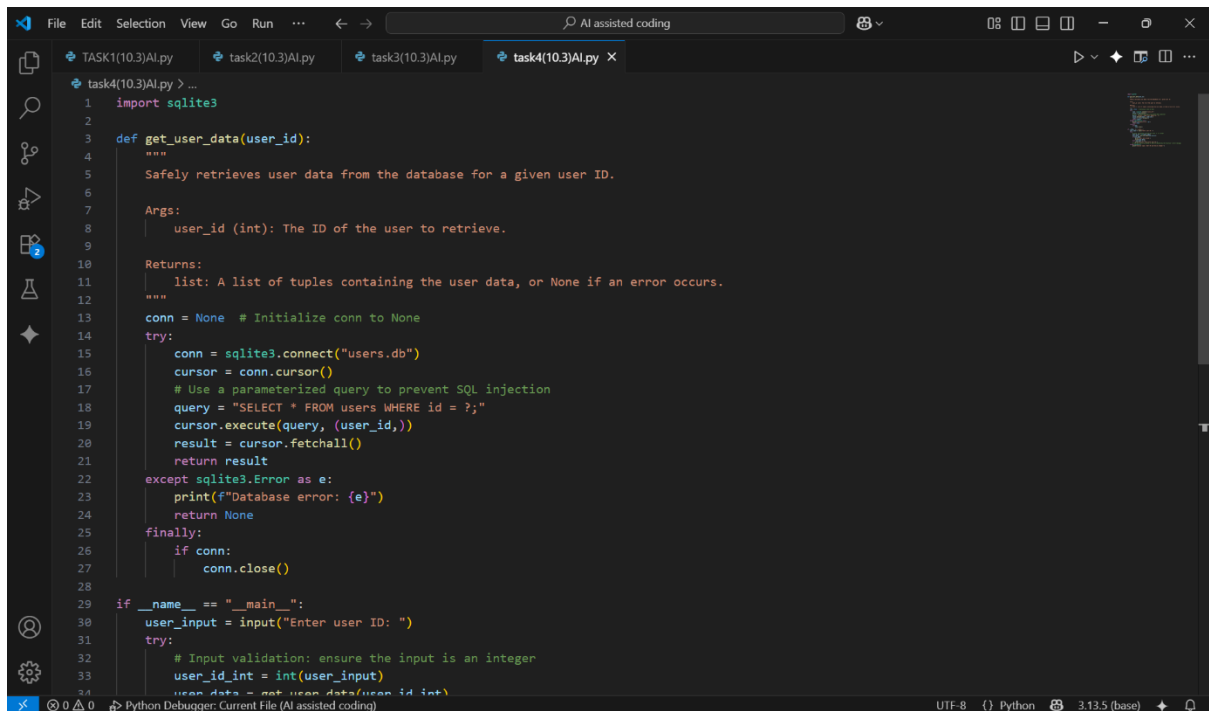
```
user_input = input("Enter user ID: ")
print(get_user_data(user_input))
```

### PROMPT:

Refactor the following Python script to fix a critical security vulnerability and add robust error handling. The current code is susceptible to SQL injection and lacks proper exception management.

```
import sqlite3
def get_user_data(user_id):
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()
    query = f"SELECT * FROM users WHERE id = {user_id};" #
    Potential SQL injection risk
    cursor.execute(query)
    result = cursor.fetchall()
    conn.close()
    return result
user_input = input("Enter user ID: ")
print(get_user_data(user_input))
```

### CODE:

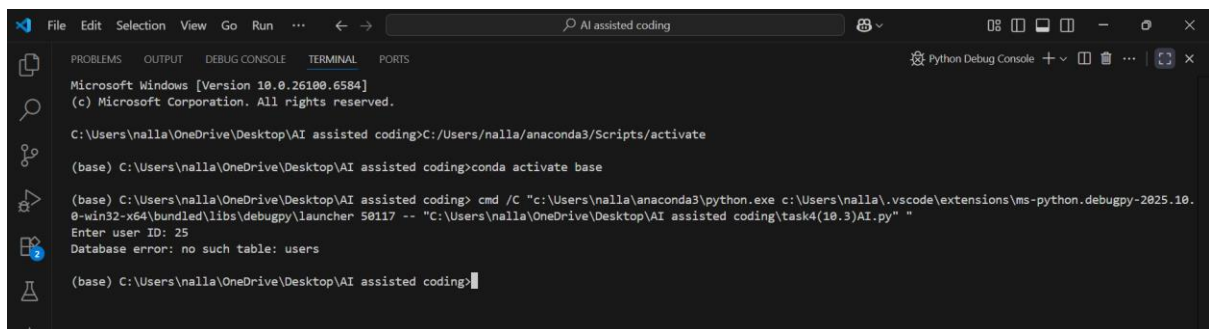


```
1  import sqlite3
2
3  def get_user_data(user_id):
4      """
5      Safely retrieves user data from the database for a given user ID.
6
7      Args:
8          user_id (int): The ID of the user to retrieve.
9
10     Returns:
11         list: A list of tuples containing the user data, or None if an error occurs.
12     """
13     conn = None # Initialize conn to None
14     try:
15         conn = sqlite3.connect("users.db")
16         cursor = conn.cursor()
17         # Use a parameterized query to prevent SQL injection
18         query = "SELECT * FROM users WHERE id = ?;"
19         cursor.execute(query, (user_id,))
20         result = cursor.fetchall()
21         return result
22     except sqlite3.Error as e:
23         print(f"Database error: {e}")
24         return None
25     finally:
26         if conn:
27             conn.close()
28
29 if __name__ == "__main__":
30     user_input = input("Enter user ID: ")
31     try:
32         # Input validation: ensure the input is an integer
33         user_id_int = int(user_input)
34         user_data = get_user_data(user_id_int)
```



```
33 user_id_int = int(user_input)
34 user_data = get_user_data(user_id_int)
35 if user_data:
36     print("User data found:")
37     print(user_data)
38 elif user_data == []:
39     print("No user found with that ID.")
40 # The case where user_data is None is handled by the function's error message
41 except ValueError:
42     print("Invalid input. User ID must be an integer.")
```

## OUTPUT:



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalla\OneDrive\Desktop\AI assisted coding>C:\Users\nalla\anaconda3\Scripts\activate

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher 50117 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\task4(10.3)AI.py" "
Enter user ID: 25
Database error: no such table: users

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>
```

## OBSERVATION:

This Python script queries a SQLite database (users.db) to retrieve user data by ID. The `get_user_data` function uses a parameterized query (`id = ?`) to prevent SQL injection and wraps operations in `try...except...finally` to handle errors and ensure the connection closes. The main block (`if __name__ == "__main__"`) prompts for an ID, validates numeric input, and gives clear feedback—showing results if found, noting when no user exists, or reporting invalid input.

## TASK 5:

Generate a review report for this messy code.

# buggy\_code\_task5.py

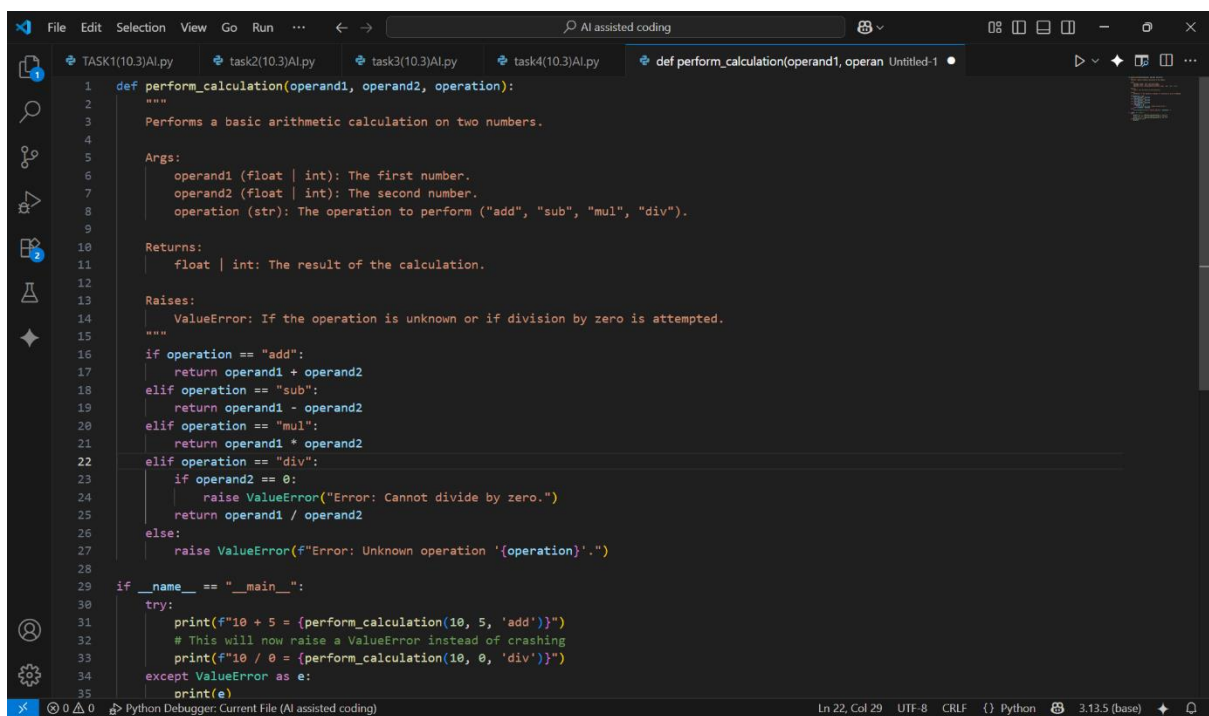
```
def calc(x,y,z):
if z=="add":
return x+y
elif z=="sub": return x-y
elif z=="mul":
return x*y
elif z=="div":
return x/y
else: print("wrong")
print(calc(10,5,"add"))
print(calc(10,0,"div"))
```

## PROMPT:

Act as an automated code review tool. Analyze the following Python script and generate a formal review report.

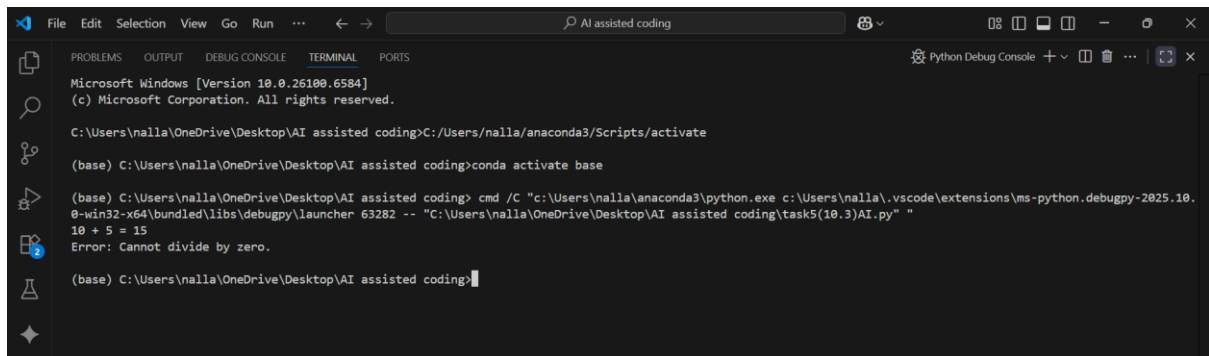
```
def calc(x,y,z):
if z=="add":
return x+y
elif z=="sub": return x-y
elif z=="mul":
return x*y
elif z=="div":
return x/y
else: print("wrong")
print(calc(10,5,"add"))
print(calc(10,0,"div"))
```

**CODE:**



```
1 def perform_calculation(operand1, operand2, operation):
2     """
3     Performs a basic arithmetic calculation on two numbers.
4
5     Args:
6         operand1 (float | int): The first number.
7         operand2 (float | int): The second number.
8         operation (str): The operation to perform ("add", "sub", "mul", "div").
9
10    Returns:
11        float | int: The result of the calculation.
12
13    Raises:
14        ValueError: If the operation is unknown or if division by zero is attempted.
15    """
16    if operation == "add":
17        return operand1 + operand2
18    elif operation == "sub":
19        return operand1 - operand2
20    elif operation == "mul":
21        return operand1 * operand2
22    elif operation == "div":
23        if operand2 == 0:
24            raise ValueError("Error: Cannot divide by zero.")
25        return operand1 / operand2
26    else:
27        raise ValueError(f"Error: Unknown operation '{operation}'.")
28
29 if __name__ == "__main__":
30     try:
31         print(f"10 + 5 = {perform_calculation(10, 5, 'add')}")
32         # This will now raise a ValueError instead of crashing
33         print(f"10 / 0 = {perform_calculation(10, 0, 'div')}")
34     except ValueError as e:
35         print(e)
```

**OUTPUT:**



```
Microsoft Windows [Version 10.0.26100.6584]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nalla\OneDrive\Desktop\AI assisted coding>C:/Users/nalla/anaconda3/Scripts/activate

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>conda activate base

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>cmd /C "c:\Users\nalla\anaconda3\python.exe c:\Users\nalla\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundle\libs\debugpy\launcher 63282 -- "C:\Users\nalla\OneDrive\Desktop\AI assisted coding\task5(10.3)AI.py" "
```

10 + 5 = 15

Error: Cannot divide by zero.

(base) C:\Users\nalla\OneDrive\Desktop\AI assisted coding>

## OBSERVATION:

This Python script defines a function `calc` that performs four basic arithmetic operations: addition, subtraction, multiplication, and division. It checks if the operation requested is valid and raises an error if it is not. Division is handled carefully by raising an error when the second number is zero to prevent a crash. In the main block, the script demonstrates how the function works by performing a valid calculation ( $10 + 5$ ) and an invalid one ( $10 / 0$ ). The results are displayed on the screen, and any errors are caught and printed in a user-friendly way.