SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
NAME:MANASA ACHINA **ENROLL NO.:2403A53043** **BATCH NO.:24BTCAICYB02**	**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Venkataramana Veeramsetty	
Instructor(s) Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	

Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week5 - Thursday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	

AssignmentNumber:10.4(Present assignment number)/**24**(Total number of assignments)

Q.No.	**Question**	***Expected Time to complete***
1	**Lab 10 – Code Review and Quality: Using AI to Improve Code Quality and Readability** **Lab Objectives** • Use AI for automated code review and quality enhancement. • Identify and fix syntax, logical, performance, and security issues in Python code. • Improve readability and maintainability through structured	Week5 - Thursday

refactoring and comments.

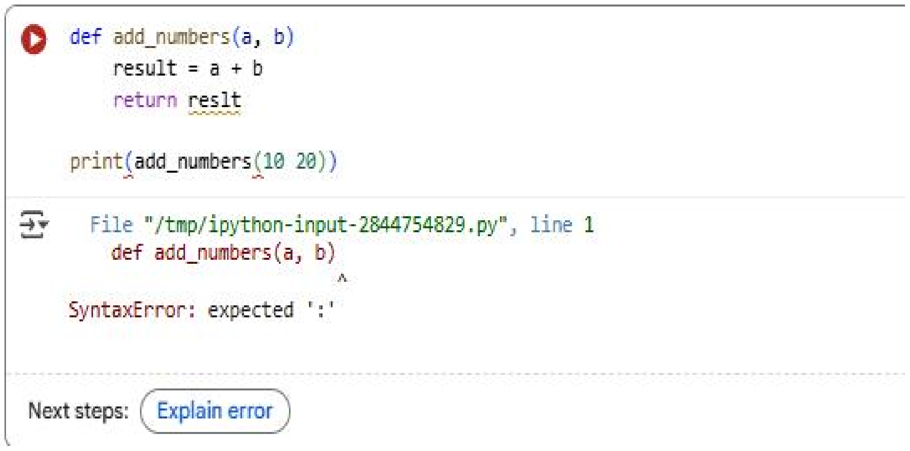
- Apply prompt engineering for targeted improvements.
- Evaluate AI-generated suggestions against PEP 8 standards and software engineering best practices

---

Task 1: Syntax and Error Detection

Task: 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))
```

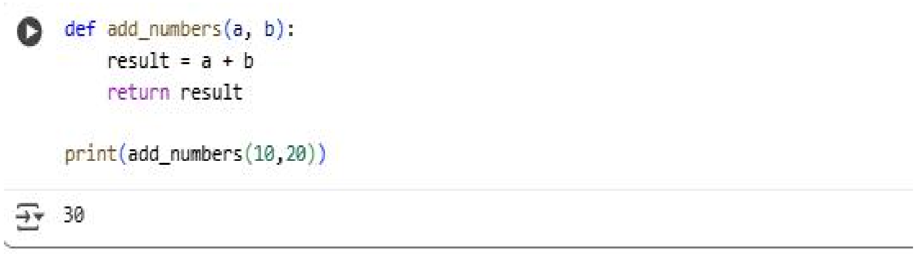


```
def add_numbers(a, b)
    result = a + b
    return reslt

print(add_numbers(10 20))
```

```
  File "/tmp/ipython-input-2844754829.py", line 1
    def add_numbers(a, b)
                         ^
SyntaxError: expected ':'
```

Expected Output:

- Corrected code with proper syntax (: after function, fixed variable name, corrected function call).



```
def add_numbers(a, b):
    result = a + b
    return result

print(add_numbers(10,20))
```

```
30
```

- **AI should explain what was fixed.**

- A colon : was added at the end of the function definition line def add_numbers(a, b). In

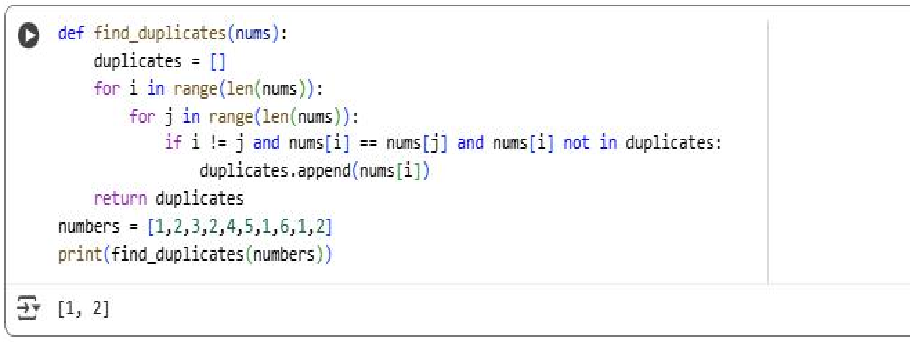
Python, function definitions must end with a colon.

- Indentation Error: The lines inside the function (result = a + b and return result) were not properly indented. In Python, indentation is crucial for defining code blocks. They should be indented with four spaces or one tab under the function definition.
- Variable Error: The variable name reslt in the return statement was misspelled. It was corrected to result to match the variable where the sum is stored.
- Syntax Error: The function call print(add_numbers(10 20)) was missing a comma between the arguments 10 and 20. A comma is required to separate arguments in a function call.

Task 2: Logical and Performance Issue Review

Task: 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))
```

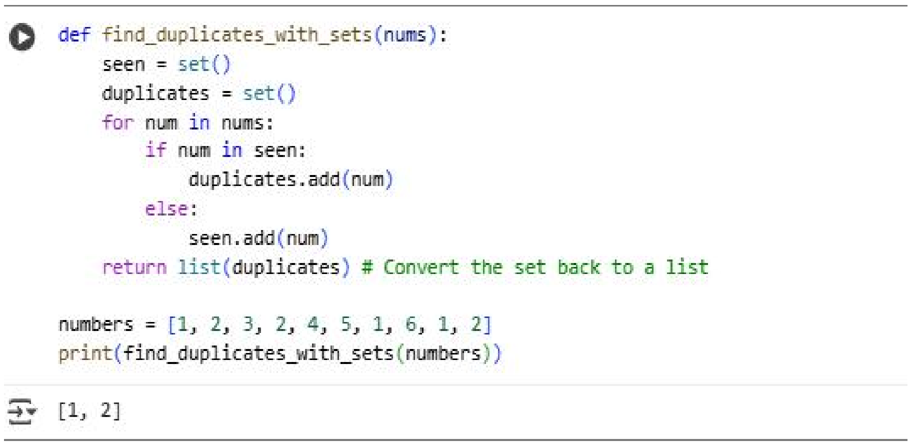


```
def find_duplicates(nums):
    duplicates = []
    for i in range(len(nums)):
        for j in range(len(nums)):
            if i != j and nums[i] == nums[j] and nums[i] not in duplicates:
                duplicates.append(nums[i])
    return duplicates
numbers = [1,2,3,2,4,5,1,6,1,2]
print(find_duplicates(numbers))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

```
[1, 2]
```

Expected Output:

- More efficient duplicate detection (e.g., using sets).


```python
def find_duplicates_with_sets(nums):
    seen = set()
    duplicates = set()
    for num in nums:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)
    return list(duplicates) # Convert the set back to a list

numbers = [1, 2, 3, 2, 4, 5, 1, 6, 1, 2]
print(find_duplicates_with_sets(numbers))
```

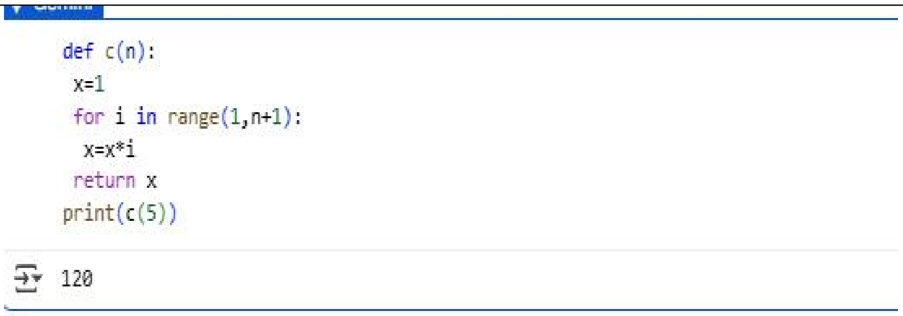
```
[1, 2]
```

- **AI should explain the optimization.**

- Initialize two sets: seen and duplicates: The seen set will keep track of all the numbers we've encountered so far. The duplicates set will store the numbers that are found to be duplicates.
- Iterate through the list once: The code goes through the nums list just one time.
- Check for duplicates using the seen set: For each number in the list, it checks if the number is already in the seen set.
- If the number is in seen, it means we've seen this number before, so it's a duplicate. We add this number to the duplicates set.
- If the number is not in seen, it means this is the first time we're seeing this number. We add it to the seen set.
- Return the duplicates: Finally, the code converts the duplicates set back into a list and returns it.

Task 3: Code Refactoring for Readability
Task: 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))
```

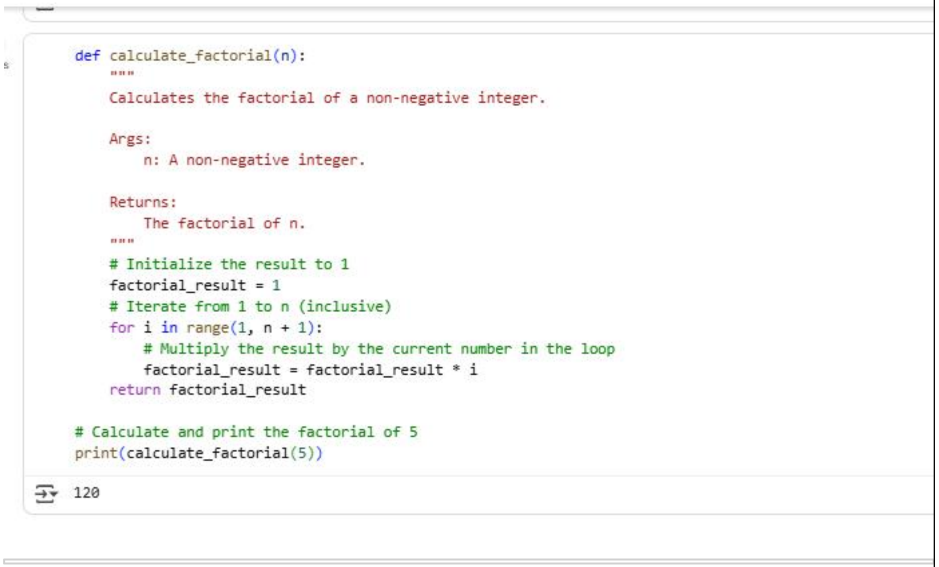


```
def c(n):
 x=1
 for i in range(1,n+1):
  x=x*i
 return x
print(c(5))
```

```
120
```

Expected Output:

Function renamed to calculate_factorial.
Proper indentation, variable naming, docstrings, and formatting.
AI should provide a more readable version.



```
def calculate_factorial(n):
    """
    Calculates the factorial of a non-negative integer.

    Args:
        n: A non-negative integer.

    Returns:
        The factorial of n.
    """
    # Initialize the result to 1
    factorial_result = 1
    # Iterate from 1 to n (inclusive)
    for i in range(1, n + 1):
        # Multiply the result by the current number in the loop
        factorial_result = factorial_result * i
    return factorial_result

# Calculate and print the factorial of 5
print(calculate_factorial(5))
```

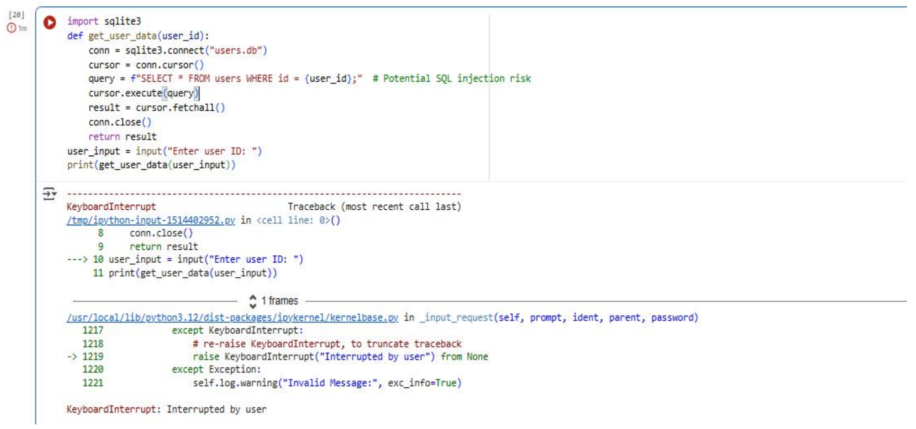
```
120
```

Task 4: Security and Error Handling Enhancement

Task: 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))
```



```
[20]
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))
```

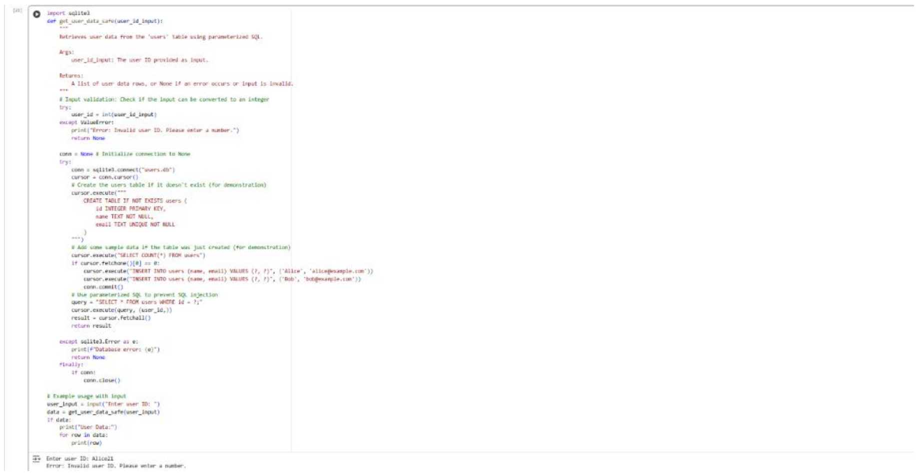
```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-1514402952.py in <cell line: 0>()
      8     conn.close()
      9     return result
---> 10 user_input = input("Enter user ID: ")
     11 print(get_user_data(user_input))

                              1 frames
/usr/local/lib/python3.12/dist-packages/ipykernel/kernelbase.py in _input_request(self, prompt, ident, parent, password)
   1217             except KeyboardInterrupt:
   1218                 # re-raise KeyboardInterrupt, to truncate traceback
-> 1219                 raise KeyboardInterrupt("Interrupted by user") from None
   1220             except Exception:
   1221                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

Expected Output:

Safe query using parameterized SQL (? placeholders).

Try-except block for database errors.

Input validation before query execution.



```
import sqlite3
def get_user_data_safe(user_id_input):
    """
    Retrieves user data from the 'users' table using parameterized SQL.

    Args:
        user_id_input: The user ID provided as input.

    Returns:
        A list of user data rows, or None if an error occurs or input is invalid.
    """
    # Input validation: Check if the input can be converted to an integer
    try:
        user_id = int(user_id_input)
    except ValueError:
        print("Error: Invalid user ID. Please enter a number.")
        return None

    conn = None # Initialize connection to None
    try:
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()
        # Create the users table if it doesn't exist (for demonstration)
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY,
                name TEXT NOT NULL,
                email TEXT UNIQUE NOT NULL
            )
        """)
        # Add some sample data if the table was just created (for demonstration)
        cursor.execute("SELECT COUNT(*) FROM users")
        if cursor.fetchone()[0] == 0:
            cursor.execute("INSERT INTO users (name, email) VALUES (?, ?)", ('Alice', 'alice@example.com'))
            cursor.execute("INSERT INTO users (name, email) VALUES (?, ?)", ('Bob', 'bob@example.com'))
            conn.commit()
        # Use parameterized SQL to prevent SQL injection
        query = "SELECT * FROM users WHERE id = ?;"
        cursor.execute(query, (user_id,))
        result = cursor.fetchall()
        return result

    except sqlite3.Error as e:
        print(f"Database error: {e}")
        return None
    finally:
        if conn:
            conn.close()

# Example usage with input
user_input = input("Enter user ID: ")
data = get_user_data_safe(user_input)
if data:
    print("User Data:")
    for row in data:
        print(row)
```

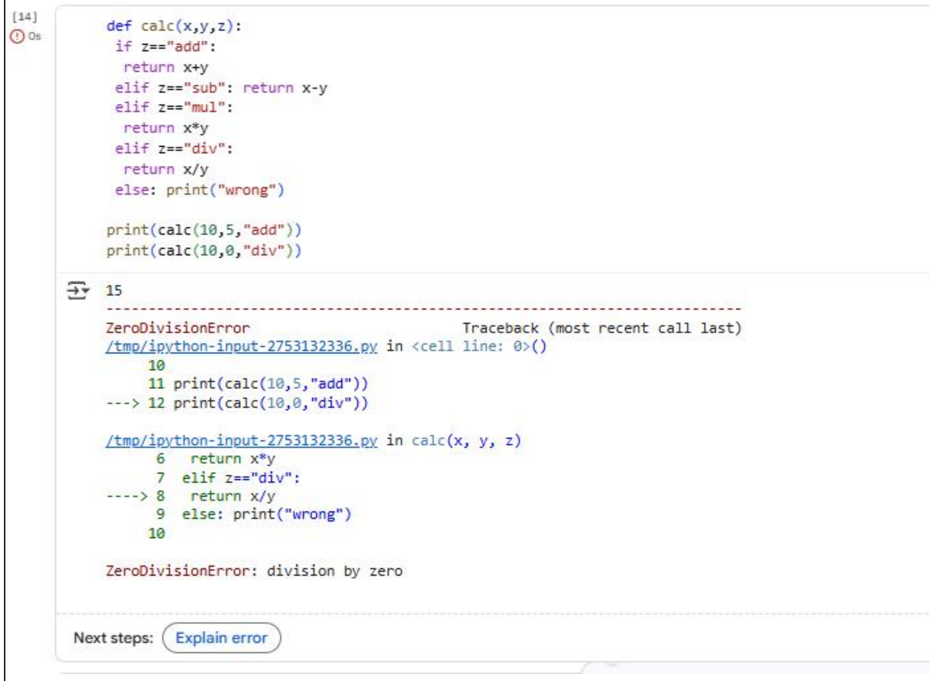
```
Enter user ID: Alice21
Error: Invalid user ID. Please enter a number.
```

Task 5: Automated Code Review Report Generation

Task: 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"))
```



```
[14]
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"))
```

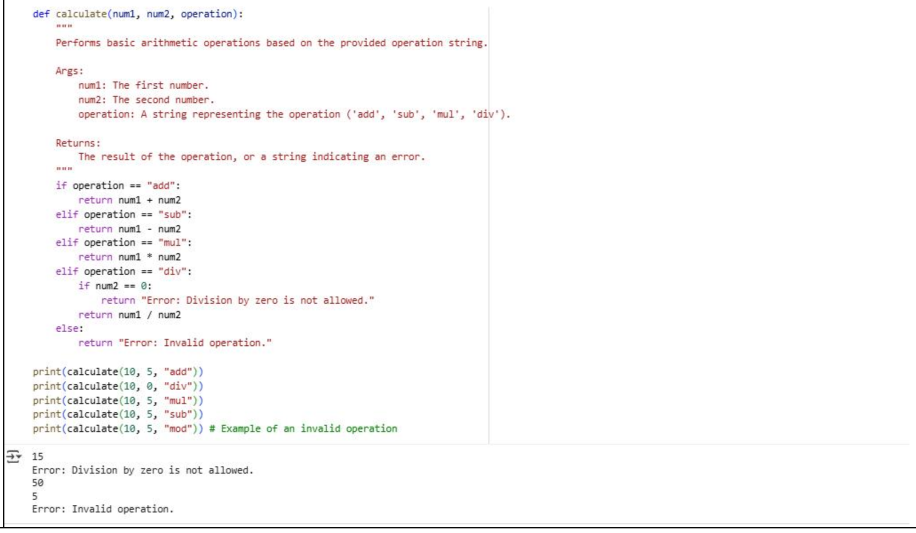
```
15
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
/tmp/ipython-input-2753132336.py in <cell line: 0>()
     10
     11 print(calc(10,5,"add"))
---> 12 print(calc(10,0,"div"))

/tmp/ipython-input-2753132336.py in calc(x, y, z)
      6   return x*y
      7  elif z=="div":
----> 8   return x/y
      9  else: print("wrong")
     10

ZeroDivisionError: division by zero
```

Expected Output:

AI-generated **review report** should mention:

- Missing docstrings
- Inconsistent formatting (indentation, inline return)

- Missing error handling for division by zero
- Non-descriptive function/variable names
- Suggestions for readability and PEP 8 compliance
-



```python
def calculate(num1, num2, operation):
    """
    Performs basic arithmetic operations based on the provided operation string.

    Args:
        num1: The first number.
        num2: The second number.
        operation: A string representing the operation ('add', 'sub', 'mul', 'div').

    Returns:
        The result of the operation, or a string indicating an error.
    """
    if operation == "add":
        return num1 + num2
    elif operation == "sub":
        return num1 - num2
    elif operation == "mul":
        return num1 * num2
    elif operation == "div":
        if num2 == 0:
            return "Error: Division by zero is not allowed."
        return num1 / num2
    else:
        return "Error: Invalid operation."

print(calculate(10, 5, "add"))
print(calculate(10, 0, "div"))
print(calculate(10, 5, "mul"))
print(calculate(10, 5, "sub"))
print(calculate(10, 5, "mod")) # Example of an invalid operation
```

```
15
Error: Division by zero is not allowed.
50
5
Error: Invalid operation.
```