Simple Calculator Tutorial

> ## Objective
>
> To build a simple calculator that performs basic arithmetic operations (addition, subtraction, multiplication, division) using functions. This exercise will help learners understand function definition, calling, and parameter passing, variable scope, code modularization, stack traces, and debugging techniques.

1. Non-Modularized Operations

1.1 Define Arithmetic Operations without Functions

Addition

```
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
result = num1 + num2
print(f"{num1} + {num2} = {result}")
```

Subtraction

```
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
result = num1 - num2
print(f"{num1} - {num2} = {result}")
```

Multiplication

```
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
result = num1 * num2
print(f"{num1} * {num2} = {result}")
```

Division

```
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
if num2 == 0:
    print("Error! Division by zero.")
else:
    result = num1 / num2
    print(f"{num1} / {num2} = {result}")
```

2. Modularized Operations with Functions

2.1 Define Functions for Arithmetic Operations

Addition Function

```
def add(a: int, b: int) -> int:
    """
    Adds two integers and returns the result.

    Parameters:
    a (int): The first integer to add.
    b (int): The second integer to add.

    Returns:
    int: The sum of the two integers.
    """
    return a + b
```

Subtraction Function

```
def subtract(a: int, b: int) -> int:
    """
    Subtracts the second integer from the first and returns t

    Parameters:
    a (int): The integer to subtract from.
    b (int): The integer to subtract.

    Returns:
    int: The result of the subtraction.
```

```
    """
    return a - b
```

Multiplication Function

```
def multiply(a: int, b: int) -> int:
    """
    Multiplies two integers and returns the result.

    Parameters:
    a (int): The first integer to multiply.
    b (int): The second integer to multiply.

    Returns:
    int: The product of the two integers.
    """
    return a * b
```

Division Function

```
def divide(a: int, b: int) -> float:
    """
    Divides the first integer by the second and returns the r

    Parameters:
    a (int): The numerator.
    b (int): The denominator.

    Returns:
    float: The result of the division.
            If the denominator is zero, returns an error mess
    """
    if b == 0:
        return "Error! Division by zero."
    return a / b
```

2.2 Keyword Arguments

Functions can accept arguments in two ways: positional or keyword arguments. Keyword arguments allow specifying parameters by their names, improving readability and flexibility.

Example of Keyword Arguments

```
def greet(name: str, message: str = "Hello") -> str:
    """
    Greets a person with a specified message.

    Parameters:
    name (str): The name of the person to greet.
    message (str): The greeting message. Defaults to "Hello"

    Returns:
    str: A formatted greeting message.
    """
    return f"{message}, {name}!"

print(greet("Alice"))  # Positional argument
print(greet(name="Bob", message="Hi"))  # Keyword arguments
print(greet(message="Welcome", name="Charlie"))  # Order flex
```

Task for Learners: Modify the calculator functions to accept arguments as keywords, e.g., `add(a=5, b=10)`.

3. Variable Scope (Local and Global)

3.1 Use Global Variables for Storing Results

In Python, the global keyword is used within a function to indicate that you intend to modify a variable that exists in the global scope (i.e., outside of any function). If you only need to read a global variable's value within a function, you don't need global. However, if you need to change the global variable's value, using global is essential; otherwise, Python will create a new local variable with the same name within the function's scope, leaving the global variable unchanged. Therefore, global is crucial for functions that need to directly alter global variables, but it should be used judiciously to avoid making code harder to understand and maintain.

```
global_var = 10

def read_global():
    print(global_var)  # Reads the global variable - no 'glob
```

```
def modify_global():
    global global_var
    global_var = 20  # Modifies the global variable - 'global

def try_modify_without_global():
    global_var = 30 # Creates a local variable - the global
    print(global_var)

read_global() # Output: 10
modify_global()
read_global() # Output: 20
try_modify_without_global() # Output: 30
read_global() # Output: 20
```

4. Code Modularization with Functions

4.1 Main Function to Manage Calculator Operations

```
def calculator():
    print("Simple Calculator")
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")

    choice = input("Enter choice (1/2/3/4): ")

    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))

    if choice == '1':
        print(f"{num1} + {num2} = {add(num1, num2)}")
    elif choice == '2':
        print(f"{num1} - {num2} = {subtract(num1, num2)}")
    elif choice == '3':
        print(f"{num1} * {num2} = {multiply(num1, num2)}")
    elif choice == '4':
        print(f"{num1} / {num2} = {divide(num1, num2)}")
    else:
        print("Invalid Input")
```

5. Stack Traces and Error Interpretation

5.1 Demonstrate a Stack Trace with Error Handling

```
def divide(a, b):
    print(f"Dividing {a} by {b}")
    if b == 0:
        return "Error! Division by zero."
    return a / b

result = divide(10, 0)
print(result)
```

6. Debugging Process and Techniques

6.1 Debugging Using Print Statements

```
def divide(a, b):
    print(f"Dividing {a} by {b}")
    if b == 0:
        return "Error! Division by zero."
    return a / b

result = divide(10, 0)
print(result)
```

6.2 Using a Debugger Tool

Use the debugger tool in your IDE to set breakpoints at the function definitions and the main function. Step through the code to observe the flow of execution and variable values.

Final Integration

Putting It All Together

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
```

```
def multiply(a, b):
    return a * b

def divide(a, b):
    print(f"Dividing {a} by {b}")
    if b == 0:
        return "Error! Division by zero."
    return a / b

def calculator():
    print("Simple Calculator")
    print("Select operation:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")

    choice = input("Enter choice (1/2/3/4): ")

    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))

    if choice == '1':
        print(f"{num1} + {num2} = {add(num1, num2)}")
    elif choice == '2':
        print(f"{num1} - {num2} = {subtract(num1, num2)}")
    elif choice == '3':
        print(f"{num1} * {num2} = {multiply(num1, num2)}")
    elif choice == '4':
        print(f"{num1} / {num2} = {divide(num1, num2)}")
    else:
        print("Invalid Input")

calculator()
```