Scope Questions:

1. What will be the output of the following code?

```
x = 10

def modify_x():
    x = 5
    print(x)

modify_x()
print(x)
```

2. Identify the scope of the variable `result` in the following code:

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

print(calculate(2, 3))
```

3. What will happen if you try to access `local_var` outside the function?

```
def show():
    local_var = "I'm local"
    print(local_var)

show()
# print(local_var)  # Uncomment this line and see the result
```

4. Consider the following code. What is the scope of `y`?

```
y = 7

def multiply_by_y(z):
    return z * y

print(multiply_by_y(5))
```

5. Will this code raise an error? If so, why?

```
def example():
     print(global_var)

global_var = "Available globally"
example()
```

6. Predict the output and explain the scope of variables:

```
num = 50
def change_num():
    global num
    num = 25

change_num()
print(num)
```

7. Which variables in this code snippet are local and which are global?

```
name = "Alice"

def greet():
    name = "Bob"
    print("Hello, " + name)

greet()
print("Hello, " + name)
```

8. How can you modify this function to access the global variable `counter` instead of the local one?

```
counter = 10

def update_counter():
    counter = 5
    return counter

print(update_counter())
print(counter)
```

9. What will be printed by this code?

```
total = 100

def calculate():
    total = 200
    return total

print(calculate())
print(total)
```

10. Why does the following code raise an error, and how can it be fixed?

```
def outer_func():
     x = "outer"

def inner_func():
    print(x)
    inner_func()

outer_func()
print(x)  # Fix this line to prevent the error
```

11. What will happen if we try to access `num` inside the function `test`?

```
num = 10

def test():
    print(num)

test()
```

12. Explain the difference between `local_num` and `global_num` in this code:

```
global_num = 100
def display():
    local_num = 50
    print(local_num)

display()
print(global_num)
```

13. What will be the output and why?

```
value = 10

def add_value():
    value = 20
    value += 5
    return value

print(add_value())
print(value)
```

14. Identify and correct the error in the code related to variable scope:

```
def outer():
     num = 10
     return inner()
```

```
    def inner():
        num += 5
        return num

print(outer())
```

15. What is the scope of the variable `msg` in the code below?

```
def display_message():
    msg = "Hello"
    print(msg)

display_message()
print(msg)
```

Function Call Stack Questions:

1. Trace the function calls and predict the output:

```
def first():
     return "First"

def second():
    return first() + " and Second"

print(second())
```

2. What will be printed by this code, considering the function call stack?

```
def alpha():
     return "A"

def beta():
    return alpha() + " B"
```

```
def gamma():
    return beta() + " C"

print(gamma())
```

3. Explain how the function call stack operates in this code:

```
def outer(x):
     y = inner(x + 2)
     return y + 3

def inner(z):
    return z * 2

print(outer(5))
```

4. What is the role of the return value in the function call stack here?

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

def add_and_multiply(x, y):
    sum_xy = x + y
    return multiply(sum_xy, y)

result = add_and_multiply(3, 4)
print(result)
```

5. Identify the flow of control in the following code:

```
def start():
     return mid()

def mid():
    return end()

def end():
    return "End"
```

```
print(start())
```

6. Analyze the function calls and their order:

```
def process(data):
     return "Processing " + data

def start_process():
    info = "Data"
    return process(info)

print(start_process())
```

7. Predict the output and explain the call stack behavior:

```
def func1():
     return "Func1"

def func2():
    return func1() + " Func2”

def func3():
    return func2() + " Func3"

print(func3())
```

8. What will the final value of `result` be, considering the function calls?

```
def step1():
     return 5

def step2(val):
    return val + 10

def calculate():
    return step2(step1())
```

```
result = calculate()
print(result)
```

9. How does the function call stack handle the following code?

```
def funcA():
     return "A"

def funcB():
    return funcA() + " B"

def funcC():
    return funcB() + " C"

print(funcC())
```

10. Trace the function calls and predict the final output:

```
def first_step(x):
     return x * 2

def second_step(y):
    return y + 3

def final_step(z):
    return first_step(z) + second_step(z)

print(final_step(4))
```

11. Analyze the flow of control and output in this nested function call:

```
def level_one():
     return "Level 1"

def level_two():
    return level_one() + " Level 2"
```

```
def level_three():
    return level_two() + " Level 3"

print(level_three())
```

12. Explain how Python handles the return values in this call stack:

```
def alpha(num):
     return num + 1

def beta(num):
    return alpha(num) * 2

def gamma(num):
    return beta(num) - 3

print(gamma(4))
```

13. Predict the output and explain the function call sequence:

```
def start(a):
     return middle(a) + 1

def middle(b):
    return end(b * 2)

def end(c):
    return c - 3

print(start(3))
```

14. What happens when functions are called in this manner?

```
def first(a):
     return a * 2
def second(b):
    return first(b + 1)
```

```
def third(c):
    return second(c) + 5

print(third(4))
```

15. Describe the call stack and predict the result of the following code:

```
def main(val):
    return part1(val) + part2(val)

def part1(x):
    return x * 3

def part2(y):
    return y + 7


print(main(2))
```