## 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)

Output: 5
        10
```

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))

ans: local scope
```

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

Output: It throughs an error called Nameerror, local_var is not
defined.
```

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))
```

ans: Global scope

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

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

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

ans: It  doesn’t raise any error. It will print “Available globally”

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

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

change_num()
print(num)
```

output: 25

scope: Global

Inside this function the
num is globally assigned. And num is assigned 25 so the num
gets updated globally.

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)

ans: local - name(name = “Bob)
     global - name(name = “Alice)
```

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)
```

```
ans: By commenting the line 4(counter = 5)

or

By writing  global counter inside the function
```

9. What will be printed by this code?

```
total = 100

def
    calculate
    ():total
    = 200
    return
    total

print(calculate
())print(total)

output:
200(local)

100(global)
```

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
```

ans: in last line “print(x)”, Name error, x is not defined. If we initialize the x value globally then it won’t get any error.

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

```
num = 10
def test():
     print(num)
test()
```

ans : If we try to access num inside the function, it will take the value of num. And in this code it will print num value. (i.e 10)

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)
```

ans: local_num - It is inside the function and we can access it only inside the function. local_num assigned a value of 50 when we print the local_num, 50 will print on the consol.

Global_num - It declared outside the function, we can access it any where in the code.

13. What will be the output and why?

```
value = 10

def add_value():
     value = 20    # Local variable
     value += 5    # value = value + 5 = 25
     return value
print(add_value()) # prints 25
print(value)       # it will print Global variable value 10
```

```
output: 25
        10
```

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

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

def inner():
     num = 3
     num += 5
     return num
print(outer())
```

Output: 8  (declared a num value inside inner() function)

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

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

ans: local scope

**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())
```

output: first and second

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

```
def alpha():
     return "A"
def beta():
     return alpha() + " B"  # returns “A” + “B”
def gamma():
     return beta() + " C"    # returns “A” + “B” + “C”
print(gamma())
```

output: A B C

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))
```

explation:

x = 5
y = inner(7)
inner(7) = 7*2 = 14
return y+3 = 14 + 3 = 17
print(outer(5)) = 17

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)
```

ans: it will returns the value and we can access anywhere in the code when the function is called.

Output: 28

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

```
def start():
     return mid()
def mid():
     return end()
def end():
     return "End"
print(start())
```

ans: start() --→ mid() --→ end()

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())
```

```
Ans: process(data)
     Start_process():
     Info = "Data"
     Process(Data)
```

Returns – Processing Data
Print(start_process()) – o/p: Processin Data

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())
```

output: Func1 Func2 Func3
explanation: fun1() – returns "Func1"
            In func2()
                  Func1()- ("Func1") + "Func2" – returns "Func1" + "Func2"
            In func3()
                  Func2 – ("Func1" + "Func2") + "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)
```

output: 15

step2(step1))= step2(5) = 5+10 = 15

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())
```

```
output: A B C
```

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

```
def first_step(x):
      return x * 2   # 4*2=8

def second_step(y):
      return y + 3   # 4+3=8

def final_step(z):
      return first_step(z) + second_step(z)   # 8 + 7

print(final_step(4))
```

```
output: 15
```

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"    #”Level 1” + “Level 2”

def level_three():
      return level_two() + " Level 3"   #”leval 1” + “Level 2” +
      “Level 3”

print(level_three())
```

```
output: Level 1 Level 2 Level 3
```

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))
```

```
ans: gamma(4) -> beta(4) - 3
      beta(4) - > alpha(4) * 2
      alpha(4) - > 4 + 1
      beta(4) = 5 * 2 = 10
      gamma(4) = 10 - 3 = 7

output: 7
```

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))
```

```
ans:
start(3) -> middle(3) + 1
middle(3) -> end(3*2) -> end(6)
end(6) -> 6 -3 = 3
middle(3) -> 3
start(3) -> 3 + 1 = 4
Output: 4
```

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))
```

```
ans:
third(4) -> second(4) + 5
second(4) -> first(4+1) + 5
first(5) - > 5 * 2 = 10
second(4) - > 10
third(4) -> 15
Output: 15
```

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))
```

```
Ans:
main(2) -> part1(2) + part2(2)
part1(2) -> 2 *3 = 6
part2(2) - > 2 + 7 = 9
main(2) -> 6 + 9 = 15
Output = 15
```

**Function Call Stack Questions:**

1. What is the final value of result after the series of function calls?

```
def add(a, b):
      return a + b
def multiply(a, b):
      return a * b
result = multiply(add(2, 3), add(4, 5))
print(result)
```

```
Output:45
```

Explanation: `add(2, 3)` returns 5, `add(4, 5)` returns 9, and `multiply(5, 9)` returns 45.

2. Explain the order of operations in the call stack and predict the output:

```
def calculate(a, b):
      return (a + b) * 2
def double_sum(x, y):
      return calculate(x, y) + calculate(y, x)
result = double_sum(3, 4)
print(result)
```

```
Output:28
```

3. Predict the output of the following code and explain the function call sequence:

```
def increment(a):
      return a + 1
def decrement(b):
      return b - 1
def process(value):
      return increment(value) + decrement(value)
print(process(10))
```

```
Output: 20
```

Explanation:
- `increment(10)` returns 11.
- `decrement(10)` returns 9.
- `process(10)` returns `11 + 9`, which is equals to 20.

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

```
def calculate_area(length, width):
      return length * width
def perimeter(length, width):
      return 2 * (length + width)
def display_calculations(length, width):
      area = calculate_area(length, width)
      peri = perimeter(length, width)
      return f"Area: {area}, Perimeter: {peri}"
print(display_calculations(5, 3))
```

```
Output:   Area: 15, Perimeter: 16

Explanation:
- `calculate_area(5, 3)` returns 15.
- `perimeter(5, 3)` returns 16.
- The formatted string displays these values.
```

5. Trace the function calls and predict the output:

```
def func1(x):
      return x * 2
def func2(y):
      return func1(y) + 3
def func3(z):
      return func2(z) - 1
print(func3(4))
```

```
Output: 10

Explanation:
- `func3(4)` calls `func2(4)`, which returns `func1(4) + 3 = 8 + 3 =
11`.
- `func3(4)` then subtracts 1 from this result, giving `11 - 1 =
10`.
```

6. What will be the result of the following code? Explain the flow of control:

```
def operation1(a, b):
      return a - b
def operation2(c, d):
      return operation1(c, d) + operation1(d, c)
result = operation2(7, 5)
print(result)
```

```
Output: 0
```

```
Explanation:
- `operation1(7, 5)` returns `7 - 5 = 2`.
- `operation1(5, 7)` returns `5 - 7 = -2`.
- `operation2(7, 5)` returns `2 + (-2) = 0`.
```

7. Explain how the return values are managed in the function call stack:

```
def step1(x):
      return x * 3
def step2(y):
      return y + 4
def step3(z):
      return z - 2
result = step3(step2(step1(2)))
print(result)
```

```
Output:8

Explanation:
- `step1(2)` returns `6`.
- `step2(6)` returns `10`.
- `step3(10)` returns `8`.
```

8. Determine the output based on the order of function calls:

```
def alpha(a):
      return a + 1
def beta(b):
      return b * 2
def gamma(c):
      return c - 3
result = gamma(beta(alpha(3)))
print(result)
```

```
Output: 5

Explanation:
- `alpha(3)` returns 4.
- `beta(4)` returns 8.
- `gamma(8)` returns `8 - 3 = 5`.
```

9. What happens when the following functions are called in sequence?

```
def add_five(x):
      return x + 5
def subtract_three(y):
      return y - 3
def combine_operations(z):
      return subtract_three(add_five(z))
print(combine_operations(10))
```

```
Output: 12

Explanation:
- `add_five(10)` returns 15.
- `subtract_three(15)` returns `15 - 3 = 12`.
```

10. 10. How does the function call stack handle multiple return values?

```
def operation1(a):
      return a * 2
def operation2(b):
      return b + 5
def calculate_result(c):
      return operation1(c) + operation2(c)
print(calculate_result(3))
```

```
Output: 14

Explanation:
- `operation1(3)` returns `3 * 2 = 6`.
- `operation2(3)` returns `3 + 5 = 8`.
- `calculate_result(3)` returns `6 + 8 = 14`.
```

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

```
def first(x):
      return x + 2
def second(y):
      return y * 3
def third(z):
      return second(first(z))
result = third(4)
print(result)
```

```
Output: 18

Explanation:
- `first(4)` returns `4 + 2 = 6`.
- `second(6)` returns `6 * 3 = 18`.
```

12. What is the final value of output after these function calls?

```
def square(x):
     return x * x
def double(y):
     return y + y
def combine(x, y):
     return square(x) + double(y)
output = combine(2, 3)
print(output)
```

```
Output: 10

Explanation:
- `square(2)` returns `2 * 2 = 4`.
- `double(3)` returns `3 + 3 = 6`.
- `combine(2, 3)` returns `4 + 6 = 10`.
```

13. 13. Explain how Python handles nested function calls in this scenario:

```
def calculate(a, b):
     return (a + b) * (a - b)
def process(x, y):
     return calculate(x, y) + calculate(y, x)
print(process(5, 2))
```

```
Output: 0

Explanation:
- `calculate(5, 2)` returns `(5 + 2) * (5 - 2) = 7 * 3 = 21`.
- `calculate(2, 5)` returns `(2 + 5) * (2 - 5) = 7 * (-3) = -21`.
- `process(5, 2)` returns `21 + (-21) = 0`.
```

14. Determine the output by following the function call stack:

```
def start(a):
     return a + 5
def middle(b):
     return start(b * 2)
def end(c):
     return middle(c) - 3
print(end(4))
```

```
Output: 10
```

15. Analyze the code and predict the output:

```
def operation1(a):
     return a * 3
def operation2(b):
     return b + 7
def final_result(c):
     return operation1(c) - operation2(c)
result = final_result(5)
print(result)
```

```
Output: 3
```

**Scope and function call trace V2:**

**Scope Questions:**

1. What will be the output of the following code? Explain the scope of x.

```
x = 10
def modify():
    global x
    x = x + 5
    return x
print(modify())
print(x)
```

```
Output:15
       15
```

Explanation: The `global x` statement inside the `modify` function means that `x` refers to
the global variable, not a local one. The function modifies the global `x` by adding 5 to it.

2. Consider the following code. What is the value of total after the function call?

```
total = 100
def update_total(value):
    global total
    total = total + value
update_total(50)
print(total)
```

```
Output:
- 150
```

Explanation: The `total` variable is global, and the function `update_total` modifies it by adding 50 to it, making the total value 150.

3. Explain why the code below results in a NameError. How can it be fixed?

```
def increment_value(a):
    b = 2
    a = a + b
    return a
print(a)
print(increment_value(5))
```

Error: NameError: `a` is not defined before the first `print`.
Fix: To avoid the error, declare a outside function.

4. What will be printed by the code? Discuss the scope of message.

```
def show_message():
      message = "Hello, World!"
      return message
print(show_message())
print(message)
```

```
Output:
- Hello, World!
- NameError (on `print(message)`)
```

Explanation: `message` is local to the `show_message` function. Trying to access it outside causes a NameError.

5. Analyze the following code. What happens when the function calculate() is called?

```
result = 20
def calculate():
      result = 10
      result += 5
      return result
print(calculate())
print(result)
```

```
Output:15
       20
```

Explanation: `result` inside `calculate` is a local variable, so it doesn’t affect the global `result`.

6. What is the output of the following code? Why?

```
z = 5
def process():
      z = 10
      return z + 3
print(process())
print(z)
```

```
Output:13
       5
```

Explanation: `z` inside `process` is local and doesn’t affect the global `z`. The global `z` remains unchanged.

7. Why does the code produce different outputs inside and outside the function?

```
data = 42
def change_data():
     global data
     data = 24
     return data
print(change_data())
print(data)
```

```
Output:24
       24
```

Explanation: The `global data` allows the function to modify the global variable `data`.

8. How does the global scope interact with the function's local scope in this code?

```
price = 100
def apply_discount():
     global price
     discount = 20
     price -= discount
apply_discount()
print(price)
```

```
Output:80
```

Explanation: The global variable `price` is reduced by the value of `discount` (20), so the global `price` becomes 80.

9. Determine the final value of value after the function calls:

```
value = 10
def multiply_by_two():
     global value
     value = value * 2
multiply_by_two()
multiply_by_two()
print(value)
```

```
Output:40
```

Explanation: The first call to `multiply_by_two` changes `value` to 20, and the second call doubles it to 40.

10. 10. Explain the error in the code and how to fix it.

```
def calculate_area():
      length = 10
      width = 5
      return length * width
print(length)
print(calculate_area())
```

```
Error: NameError: `length` is not defined.
Fix: Move the `print(length)` inside the function where `length` is
defined.
```

11. Why does the following code result in different outputs when the variable is accessed inside and outside the function?

```
status = "active"
def deactivate():
      global status
      status = "inactive"
      return status
print(deactivate())
print(status)
```

```
Output:inactive
        Inactive

Explanation: The global variable `status` is modified inside the
`deactivate` function.
```

12. Predict the output and explain the scope of the variable x:

```
x = 50
def update_x():
      x = 25
      return x
print(update_x())
print(x)
```

```
Output:25
        50
Explanation: The function’s local `x` does not affect the global
`x`, so the global `x` remains 50.
```

13. What is the final value of balance after the function call?

```
balance = 1000
def withdraw(amount):
     global balance
     balance -= amount
withdraw(200)
print(balance)
```

```
Output:800
```

Explanation: The `withdraw` function modifies the global `balance`, reducing it by 200.

14. Analyze the scope of counter and predict the output:

```
counter = 0
def increment_counter():
     counter = 1
     counter += 1
     return counter
print(increment_counter())
print(counter)
```

```
Output:2
       0
```

Explanation: The `counter` inside the function is local and does not affect the global `counter`.

15. Explain the concept of variable shadowing with the following code:

```
name = "Alice"
def change_name():
     name = "Bob"
     return name
print(change_name())
print(name)
```

```
Output: Bob
        Alice
```

Explanation: The local variable `name` inside `change_name` shadows the global `name