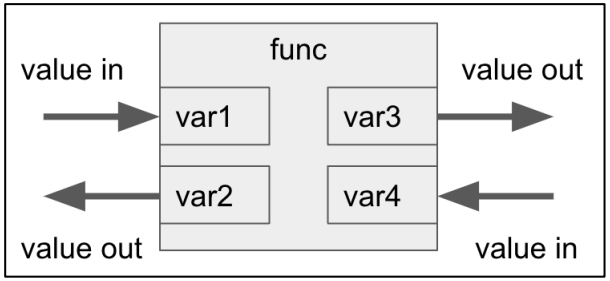
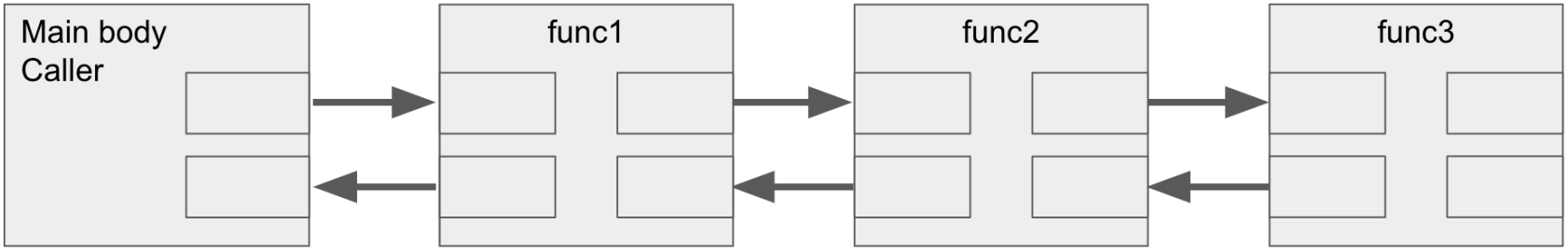
Lecture 5.1. Functions (continued)

Question 1.

Line Number	Main caller	func1			func2			func3		
	z	x1	y1	z1	x2	y2	z2	x3	y3	z3

```
1  def func3(x3):
2      y3 = x3
3      z3 = y3 - 1
4      return z3
5
6  def func2(x2):
7      y2 = func3(x2)
8      z2 = y2 - 1
9      return z2
10
11 def func1(x1):
12     y1 = func2(x1)
13     z1 = y1 - 1
14     return z1
15
16 z = func1(3)
17 print(z)
```

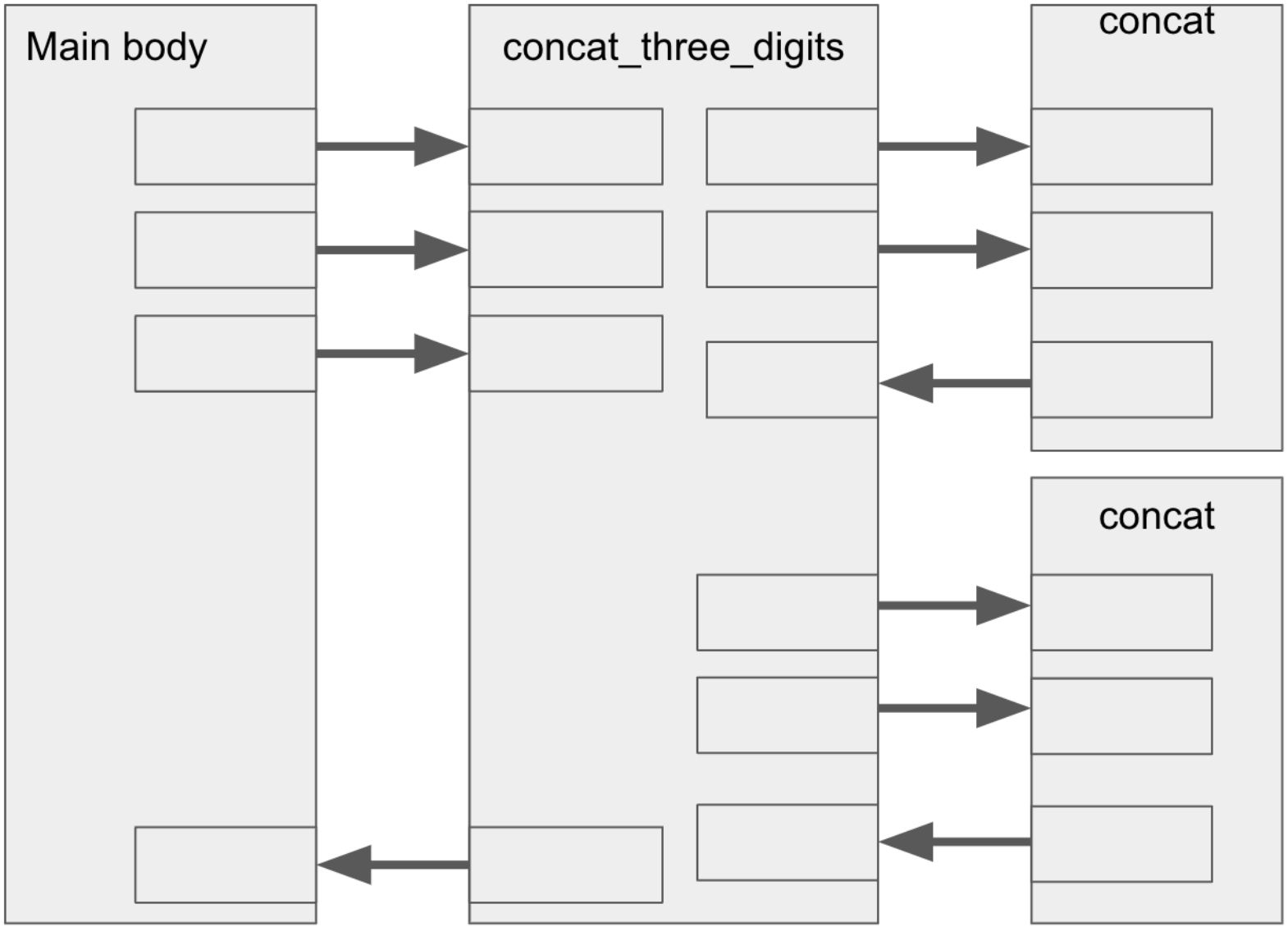




Question 2.

```
1  def concat(num, digit):
2      concatenated = num*10 + digit
3      return concatenated
4
5  def concat_three_digits(hundreds, tens, units):
6      first_two = concat(hundreds, tens)
7      all_three = concat(first_two, units)
8      return all_three
9
10 num = concat_three_digits(1, 2, 3)
11 print(num)
```

Line Number	Main caller	concat_three_digits					concat		
	num	hundreds	tens	units	first_two	all_three	num	digit	concatenated


Main body
concat_three_digits
concat
concat

Question 3

3.a. Implement `csc121_abs` function that resembles the built-in `abs` function.

3.b. Replicate `abs`'s docstring

3.c. Also write test cases for `csc121_abs`, covering as many different categories of inputs as you can think of.

Question 4

4.a. Implement `csc121_help` function resembles the basic functionality of the built-in `help` function.

Hint: Recall the three ways to access a function's documentation string.

4.b. You don't have to write test cases for `csc121_help`. However, try the following:

```
x = help(round)
print(x)
```

What gets printed, and why? How would you test or compare `help` and `csc121_help`?

Question 5

5.a. Implement `csc121_pow` function that resembles the built-in `pow` function.

5.b. Replicate `pow`'s docstring

5.c. Also write and test cases for `csc121_pow`, covering as many different categories of inputs as you can think of.