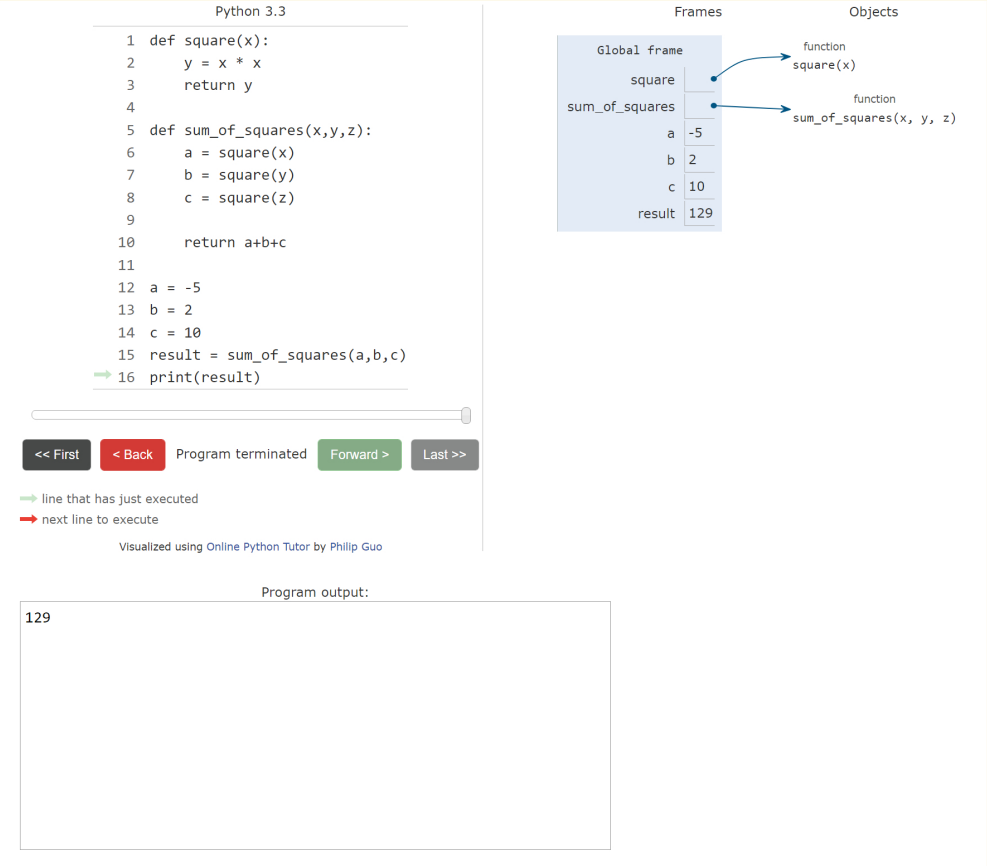
12.10. Functions can call other functions (Composition)

It is important to understand that each of the functions we write can be used and called from other functions we write. This is one of the most important ways that computer programmers take a large problem and break it down into a group of smaller problems. This process of breaking a problem into smaller subproblems is called **functional decomposition**.

Here's a simple example of functional decomposition using two functions. The first function called `square` simply computes the square of a given number. The second function called `sum_of_squares` makes use of square to compute the sum of three numbers that have been squared.



Activity: 1 -- CodeLens: (clens11_9_1)

Even though this is a pretty simple idea, in practice this example illustrates many very important Python concepts, including local and global variables along with parameter passing. Note that the body of `square` is not executed until it is called from inside the `sum_of_squares` function for the first time on line 6.

Also notice that when `square` is called (at Step 8, for example), there are two groups of local variables, one for `square` and one for `sum_of_squares`. Each group of local variables is called a **stack frame**. The variables `x`, and `y` are local variables in both functions. These are completely different variables, even though they have the same name. Each function invocation creates a new frame, and variables are looked up in that frame. Notice that at step 9, y has the value 25 is one frame and 2 in the other.

What happens when you to refer to variable y on line 3? Python looks up the value of y in the stack frame for the `square` function. If it didn't find it there, it would go look in the global frame.

Let's use composition to build up a little more useful function. Recall from the dictionaries chapter that we had a two-step process for finding the letter that appears most frequently in a text string:

1. Accumulate a dictionary with letters as keys and counts as values. See example.
2. Find the best key from that dictionary. See example.

We can make functions for each of those and then compose them into a single function that finds the most common letter.

```
15     best_key_so_far = list(ks)[0]  # Have to turn ks into a real list before using
```

```
b
```

Activity: 2 -- ActiveCode (ac_11_9_1)

Check your Understanding

1. Write two functions, one called `addit` and one called `mult`. `addit` takes one number as an input and adds 5. `mult` takes one number as an input, and multiplies that input by whatever is returned by `addit`, and then returns the result.

Save & Run | 5/14/2021, 1:48:01 PM - 2 of 2 | Show in CodeLens

```
1 def addit(a):
2     add = a+5
3     return add
4 def mult(a):
5     return a*addit(a)
6
7 addit(5)
8
```

Activity: 3 -- ActiveCode (ac11_9_1)

Result	Actual Value	Expected Value	Notes
Pass	6	6	Testing the function mult with input 1 (should be 6)
Pass	-6	-6	Testing the function mult with input -2 (should be -6)
Pass	0	0	Testing the function mult with input 0 (should be 0)
Pass	6	6	Testing the function addit with input 1 (should be 6)
Pass	3	3	Testing the function addit with input -2 (should be 3)
Pass	5	5	Testing the function addit with input 0 (should be 5)

You passed: 100.0% of the tests

You have attempted 4 of 3 activities on this page

12.9. Global Variables">
12.11. Flow of Execution Summary">
12.9. Global Variables">
✔ Completed. Well Done!
12.11. Flow of Execution Summary">Next Section - 12.11. Flow of Execution Summary

| Back to top