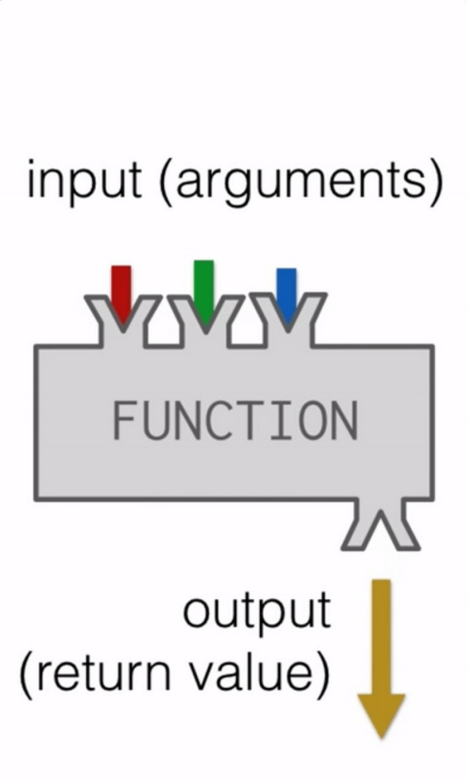
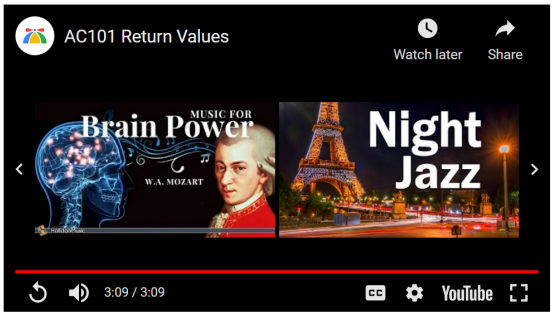
12.5. Returning a value from a function

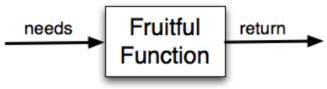


Not only can you pass a parameter value into a function, a function can also produce a value. You have already seen this in some previous functions that you have used. For example, `len` takes a list or string as a parameter value and returns a number, the length of that list or string. `range` takes an integer as a parameter value and returns a list containing all the numbers from 0 up to that parameter value.

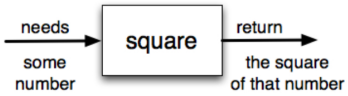
Functions that return values are sometimes called **fruitful functions**. In many other languages, a function that doesn't return a value is called a **procedure**, but we will stick here with the Python way of also calling it a function, or if we want to stress it, a *non-fruitful* function.



Activity: 1 -- Video: (goog_return_values)



How do we write our own fruitful function? Let's start by creating a very simple mathematical function that we will call `square`. The square function will take one number as a parameter and return the result of squaring that number. Here is the black-box diagram with the Python code following.



Save & Run Original - 1 of 1 Show in CodeLens

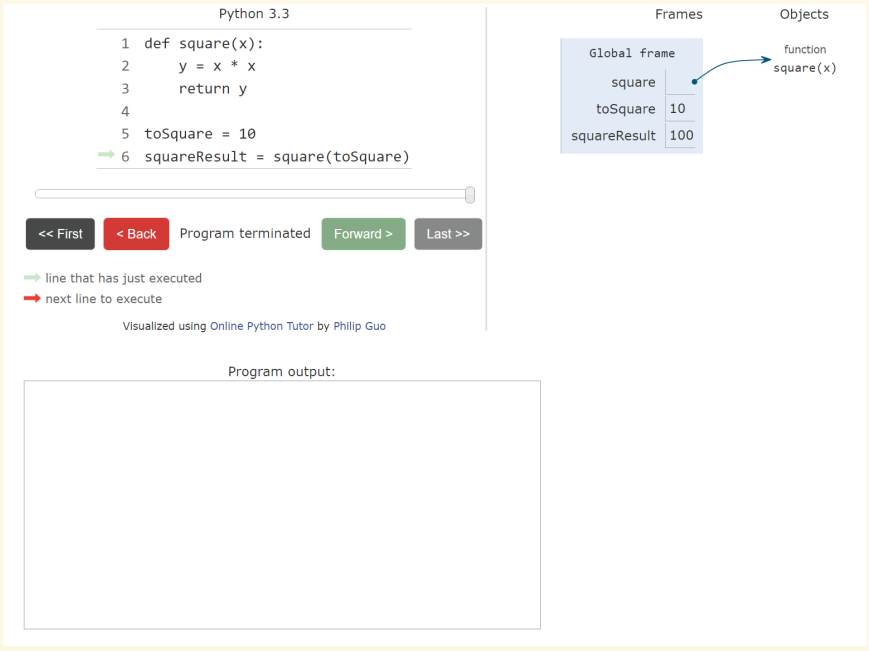
```
1 def square(x):
2     y = x * x
3     return y
4
5 toSquare = 10
6 result = square(toSquare)
7 print("The result of {} squared is {}.".format(toSquare, result))
8
```

```
The result of 10 squared is 100.
```

Activity: 2 -- ActiveCode (ac11_4_1)

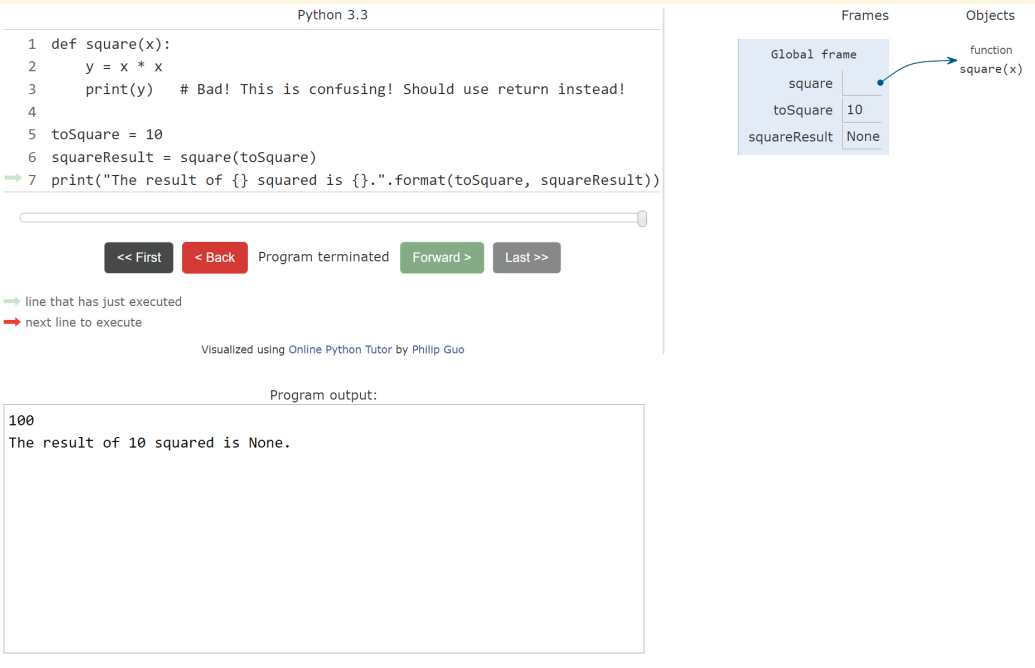
The **return** statement is followed by an expression which is evaluated. Its result is returned to the caller as the "fruit" of calling this function. Because the return statement can contain any Python expression we could have avoided creating the **temporary variable** `y` and simply used `return x*x`. Try modifying the square function above to see that this works just the same. On the other hand, using **temporary variables** like `y` in the program above makes debugging easier. These temporary variables are referred to as **local variables**.

Notice something important here. The name of the variable we pass as an argument — `toSquare` — has nothing to do with the name of the formal parameter — `x`. It is as if `x = toSquare` is executed when `square` is called. It doesn't matter what the value was named in the caller (the place where the function was invoked). Inside `square`, it's name is `x`. You can see this very clearly in codelens, where the global variables and the local variables for the square function are in separate boxes.



Activity: 3 -- CodeLens: (clens11_4_1)

There is one more aspect of function return values that should be noted. All Python functions return the special value `None` unless there is an explicit return statement with a value other than `None`. Consider the following common mistake made by beginning Python programmers. As you step through this example, pay very close attention to the return value in the local variables listing. Then look at what is printed when the function is over.



Activity: 4 -- CodeLens: (clens11_4_2)

The problem with this function is that even though it prints the value of the squared input, that value will not be returned to the place where the call was done. Instead, the value `None` will be returned. Since line 6 uses the return value as the right hand side of an assignment statement, `squareResult` will have `None` as its value and the result printed in line 7 is incorrect. Typically, functions will return values that can be printed or processed in some other way by the caller.

A return statement, once executed, immediately terminates execution of a function, even if it is not the last statement in the function. In the following code, when line 3 executes, the value 5 is returned and assigned to the variable x, then printed. Lines 4 and 5 never execute. Run the following code and try making some modifications of it to make sure you understand why "there" and 10 never print out.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1 def weird():
2     print("here")
3     return 5
4     print("there")
5     return 10
6
7 x = weird()
8 print(x)
9
```

```
here
5
```

Activity: 5 -- ActiveCode (ac11_4_2)

The fact that a return statement immediately ends execution of the code block inside a function is important to understand for writing complex programs, and it can also be very useful. The following example is a situation where you can use this to your advantage – and understanding this will help you understand other people's code better, and be able to walk through code more confidently.

Consider a situation where you want to write a function to find out, from a class attendance list, whether anyone's first name is longer than five letters, called `longer_than_five`. If there is anyone in class whose first name is longer than 5 letters, the function should return `True`. Otherwise, it should return `False`.

In this case, you'll be using conditional statements in the code that exists in the **function body**, the code block indented underneath the function definition statement (just like the code that starts with the line `print("here")` in the example above – that's the body of the function `weird`, above).

Bonus challenge for studying: After you look at the explanation below, stop looking at the code – just the description of the function above it, and try to write the code yourself! Then test it on different lists and make sure that it works. But read the explanation first, so you can be sure you have a solid grasp on these function mechanics.

First, an English plan for this new function to define called `longer_than_five`:

- You'll want to pass in a list of strings (representing people's first names) to the function.
- You'll want to iterate over all the items in the list, each of the strings.
- As soon as you get to one name that is longer than five letters, you know the function should return `True` – yes, there is at least one name longer than five letters!
- And if you go through the whole list and there was no name longer than five letters, then the function should return `False`.

Now, the code:

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1 def longer_than_five(list_of_names):
2     for name in list_of_names: # iterate over the list to look at each name
3         if len(name) > 5: # as soon as you see a name longer than 5 letters,
4             return True # then return True!
5             # If Python executes that return statement, the function is over and t
6     return False # You will only get to this line if you
7     # iterated over the whole list and did not get a name where
8     # the if expression evaluated to True, so at this point, it's correct to retur
9
10 # Here are a couple sample calls to the function with different lists of names. Tr
11
12 list1 = ["Sam","Tera","Sal","Amita"]
13 list2 = ["Rey","Ayo","Lauren","Natalie"]
14
15 print(longer_than_five(list1))
```

```
False
True
```

Activity: 6 -- ActiveCode (ac11_4_3)

So far, we have just seen return values being assigned to variables. For example, we had the line `squareResult = square(toSquare)`. As with all assignment statements, the right hand side is executed first. It invokes the `square` function, passing in a parameter value 10 (the current value of `toSquare`). That returns a value 100, which completes the evaluation of the right-hand side of the assignment. 100 is then assigned to the variable `squareResult`. In this case, the function invocation was the entire expression that was evaluated.

Function invocations, however, can also be used as part of more complicated expressions. For example, `squareResult = 2 * square(toSquare)`. In this case, the value 100 is returned and is then multiplied by 2 to produce the value 200. When python evaluates an expression like `x * 3`, it substitutes the current value of x into the expression and then does the multiplication. When python evaluates an expression like `2 * square(toSquare)`, it substitutes the return value 100 for entire function invocation and then does the multiplication.

To reiterate, when executing a line of code `squareResult = 2 * square(toSquare)`, the python interpreter does these steps:

1. It's an assignment statement, so evaluate the right-hand side expression `2 * square(toSquare)`.
2. Look up the values of the variables square and toSquare: square is a function object and toSquare is 10
3. Pass 10 as a parameter value to the function, get back the return value 100
4. Substitute 100 for square(toSquare), so that the expression now reads `2 * 100`
5. Assign 200 to variable `squareResult`

Check your understanding

func-4-1: What is wrong with the following function definition:

```
def addEm(x, y, z):
    return x+y+z
    print('the answer is', x+y+z)
```

- A. You should never use a print statement in a function definition.
- B. You should not have any statements in a function after the return statement. Once the function gets to the return statement it will immediately stop executing the function.
- C. You must calculate the value of x+y+z before you return it.
- D. A function cannot return a number.

Check me | Compare me

✔ This is a very common mistake so be sure to watch out for it when you write your code!

Activity: 7 -- Multiple Choice (question11_4_1)

func-4-2: What will the following function return?

```
def addEm(x, y, z):
    print(x+y+z)
```

- A. The value None
- B. The value of x+y+z
- C. The string 'x+y+z'

Check me | Compare me

✔ We have accidentally used print where we mean return. Therefore, the function will return the value None by default. This is a VERY COMMON mistake so watch out! This mistake is also particularly difficult to find because when you run the function the output looks the same. It is not until you try to assign its value to a variable that you can notice a difference.

Activity: 8 -- Multiple Choice (question11_4_2)

func-4-3: What will the following code output?

```
def square(x):
    y = x * x
    return y

print(square(5) + square(5))
```

- A. 25
- B. 50
- C. 25 + 25

Check me | Compare me

✔ The two return values are added together.

Activity: 9 -- Multiple Choice (question11_4_3)

func-4-4: What will the following code output?

```
def square(x):
    y = x * x
    return y

print(square(square(2)))
```

- A. 8
- B. 16
- C. Error: can't put a function invocation inside parentheses

Check me | Compare me

✔ It squares 2, yielding the value 4. 4 is then passed as a value to square again, yeilding 16.

Activity: 10 -- Multiple Choice (question11_4_4)

func-4-5: What will the following code output?

```
def cyu2(s1, s2):
    x = len(s1)
    y = len(s2)
    return x-y

z = cyu2("Yes", "no")
if z > 0:
    print("First one was longer")
else:
    print("Second one was at least as long")
```

- A. 1
- B. Yes
- C. First one was longer
- D. Second one was at least as long
- E. Error

Check me | Compare me

✔ cyu2 returns the value 1, which is assigned to z.

Activity: 11 -- Multiple Choice (question11_4_5)

func-4-6: Which will print out first, square, g, or a number?

```
def square(x):
    print("square")
    return x*x

def g(y):
    print("g")
    return y + 3

print(square(g(2)))
```

- A. square
- B. g
- C. a number

Check me | Compare me

✔ g has to be executed and return a value in order to know what paramater value to provide to x.

Activity: 12 -- Multiple Choice (question11_4_6)

func-4-7: How many lines will the following code print?

```
def show_me_numbers(list_of_ints):
    print(10)
    print("Next we'll accumulate the sum")
    accum = 0
    for num in list_of_ints:
        accum = accum + num
    return accum
    print("All done with accumulation!")

show_me_numbers([4,2,3])
```

- A. 3
- B. 2
- C. None

Check me | Compare me

✔ Yes! Two printed lines, and then the function body execution reaches a return statement.

Activity: 13 -- Multiple Choice (question11_4_7)

8. Write a function named `same` that takes a string as input, and simply returns that string.

Save & Run | 5/14/2021, 12:48:30 PM - 2 of 2 | Show in CodeLens

```
1 def same (string):
2     return string
3
```

Activity: 14 -- ActiveCode (ac11_4_4)

Result	Actual Value	Expected Value	Notes
Pass	'hello'	'hello'	Testing the same function on input 'hello'.

You passed: 100.0% of the tests

9. Write a function called `same_thing` that returns the parameter, unchanged.

Save & Run 5/14/2021, 12:48:46 PM - 2 of 2 Show in CodeLens

```
1 def same_thing(a):
2     return a
3
```

Activity: 15 -- ActiveCode (ac11_4_5)

Result	Actual Value	Expected Value	Notes
Pass	5	5	Testing the function same_thing with input 5
Pass	'Welcome'	'Welcome'	Testing the function same_thing with input 'Welcome'

You passed: 100.0% of the tests

10. Write a function called `subtract_three` that takes an integer or any number as input, and returns that number minus three.

Save & Run 5/14/2021, 12:48:56 PM - 2 of 2 Show in CodeLens

```
1 def subtract_three (a):
2     return a-3
3
```

Activity: 16 -- ActiveCode (ac11_4_6)

Result	Actual Value	Expected Value	Notes
Pass	6	6	Testing the subtract_three function on input 9.
Pass	-8	-8	Testing the subtract_three function on input -5.

You passed: 100.0% of the tests

11. Write a function called `change` that takes one number as its input and returns that number, plus 7.

Save & Run 5/14/2021, 12:49:08 PM - 2 of 2 Show in CodeLens

```
1 def change(a):
2     return a+7
3
```

Activity: 17 -- ActiveCode (ac11_4_7)

Result	Actual Value	Expected Value	Notes
Pass	12	12	Testing the function change with input 5
Pass	-3	-3	Testing the function change with input -10

You passed: 100.0% of the tests

12. Write a function named `intro` that takes a string as input. Given the string "Becky" as input, the function should return: "Hello, my name is Becky and I love SI 106."

Save & Run 5/14/2021, 12:49:32 PM - 2 of 2 Show in CodeLens

```
def intro(str1):
    return("Hello, my name is "+ str1+ " and I love SI 106.")

intro("Becky") #calling the function named intro
               #and passing the parameter Becky

```

Activity: 18 -- ActiveCode (ac11_4_8)

Result	Actual Value	Expected Value	Notes
Pass	'Hello... 106.'	'Hello... 106.'	Testing the intro function on input 'Mike'.

Expand Differences

You passed: 100.0% of the tests

13. Write a function called `s_change` that takes one string as input and returns that string, concatenated with the string " for fun.".

Save & Run 5/14/2021, 12:49:58 PM - 2 of 2 Show in CodeLens

```
def s_change(a):
    return a + " for fun."

```

Activity: 19 -- ActiveCode (ac11_4_9)

Result	Actual Value	Expected Value	Notes
Pass	'Coding for fun.'	'Coding for fun.'	Testing the function s_change with input coding
Pass	'We go... fun.'	'We go... fun.'	Testing the function s_change with input We go to the beach

Expand Differences

You passed: 100.0% of the tests

14. Write a function called `decision` that takes a string as input, and then checks the number of characters. If it has over 17 characters, return "This is a long string", if it is shorter or has 17 characters, return "This is a short string".

Save & Run 5/14/2021, 12:50:16 PM - 2 of 2 Show in CodeLens

```
def decision(s1):
    if len(s1) > 17:
        return ("This is a long string")
    else:
        return ("This is a short string")

```

Activity: 20 -- ActiveCode (ac11_4_10)

Result	Actual Value	Expected Value	Notes	
Pass	'This ...tring'	'This ...tring'	Testing the function decision with input 'Well hello dolly'	Expand Differences
Pass	'This ...tring'	'This ...tring'	Testing the function decision with input 'In olden days a glimps of stocking was looked on a something shocking but heaven knows, anything goes'	Expand Differences
Pass	'This ...tring'	'This ...tring'	Testing the function decision with input 'how do you do sir'	Expand Differences

You passed: 100.0% of the tests

You have attempted 21 of 20 activities on this page

12.4. Function Parameters">
nction Parameters">
12.6. Decoding a Function">
Completed. Well Done!
12.6. Decoding a Function">Next Section - 12.6. Decoding a Function

| Back to top