12.12. Print vs. return

Many beginning programmers find the distinction between print and return very confusing, especially since most of the illustrations of return values in intro texts like this one show the returned value from a function call by printing it, as in `print(square(g(2)))`.

The print statement is fairly easy to understand. It takes a python object and outputs a printed representation of it in the output window. You can think of the print statement as something that takes an object from the land of the program and makes it visible to the land of the human observer.

Note

Print is for people. Remember that slogan. Printing has no effect on the ongoing execution of a program. It doesn't assign a value to a variable. It doesn't return a value from a function call.

If you're confused, chances are the source of your confusion is really about returned values and the evaluation of complex expressions. A function that returns a value is producing a value for use *by the program*, in particular for use in the part of the code where the function was invoked. Remember that when a function is invoked, the function's code block is executed – all that code indented under the `def` statement gets executed, following the rules of the Python formal language for what should and should not execute as it goes. But when the function returns, control goes back to the calling location, and a return value may come back with it.

You've already seen some function calls in Chapter 1. When we told you about the function `square` that we defined, you saw that the expression `square(2)` evaluated to the integer value `4`.

That's because the `square` function *returns* a value: the square of whatever input is passed into it.

If a returned value is for use *by the program*, why did you make that function invocation to return a value? What do you use the result of the function call for? There are three possibilities.

1. **Save it for later.**

   The returned value may be:

   - Assigned to a variable. For example, `w = square(3)`
   - Put in a list. For example, `L.append(square(3))`
   - Put in a dictionary. For example, `d[3] = square(3)`

2. **Use it in a more complex expression.**

   In that case, think of the return value as replacing the entire text of the function invocation. For example, if there is a line of code `w = square(square(3) + 7) - 5`, think of the return value 9 replacing the text square(3) in that invocation, so it becomes `square(9 + 7) -5`.

3. **Print it for human consumption.**

   For example, `print(square(3))` outputs 9 to the output area. Note that, unless the return value is first saved as in possibility 1, it will be available only to the humans watching the output area, not to the program as it continues executing.

If your only purpose in running a function is to make an output visible for human consumption, there are two ways to do it. You can put one or more print statements inside the function definition and not bother to return anything from the function (the value None will be returned). In that case, invoke the function without a print statement. For example, you can have an entire line of code that reads `f(3)`. That will run the function f and throw away the return value. Of course, if square doesn't print anything out or have any side effects, it's useless to call it and do nothing with the return value. But with a function that has print statements inside it, it can be quite useful.

The other possibility is to return a value from the function and print it, as in `print(f(3))`. As you start to write larger, more complex programs, this will be more typical. Indeed the print statement will usually only be a temporary measure while you're developing the program. Eventually, you'll end up calling f and saving the return value or using it as part of a more complex expression.

You will know you've really internalized the idea of functions when you are no longer confused about the difference between print and return. Keep working at it until it makes sense to you!

Check your understanding

func-11-1: What will the following code output?

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

def g(y):
    return y + 3

def h(y):
    return square(y) + 3

print(h(2))
```

- A. 2
- B. 5
- [✓ C. 7]
- D. 25
- E. Error: y has a value but x is an unbound variable inside the square function

Check me   Compare me

✓ First square 2, then add 3.

Activity: 1 -- Multiple Choice (question11_11_1)

func-11-2: What will the following code output?

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

def g(y):
    return y + 3
```

```
def h(y):
    return square(y) + 3

print(g(h(2)))
```

- ○ A. 2
- ○ B. 5
- ○ C. 7
- ● D. 10
- ○ E. Error: you can't nest function calls

Check me Compare me

✓ h(2) returns 7, so y is bound to 7 when g is invoked.

Activity: 2 -- Multiple Choice (question11_11_2)

You have attempted 3 of 2 activities on this page