CSE231 - Lab 04

Functions, Scope, global, Exam 1

Everyone here has taken calc, right?

Functions are a topic that a lot of new programmers struggle on. Try to think of them just as the functions we know from mathematics.

$$f(x) = 2x+1$$

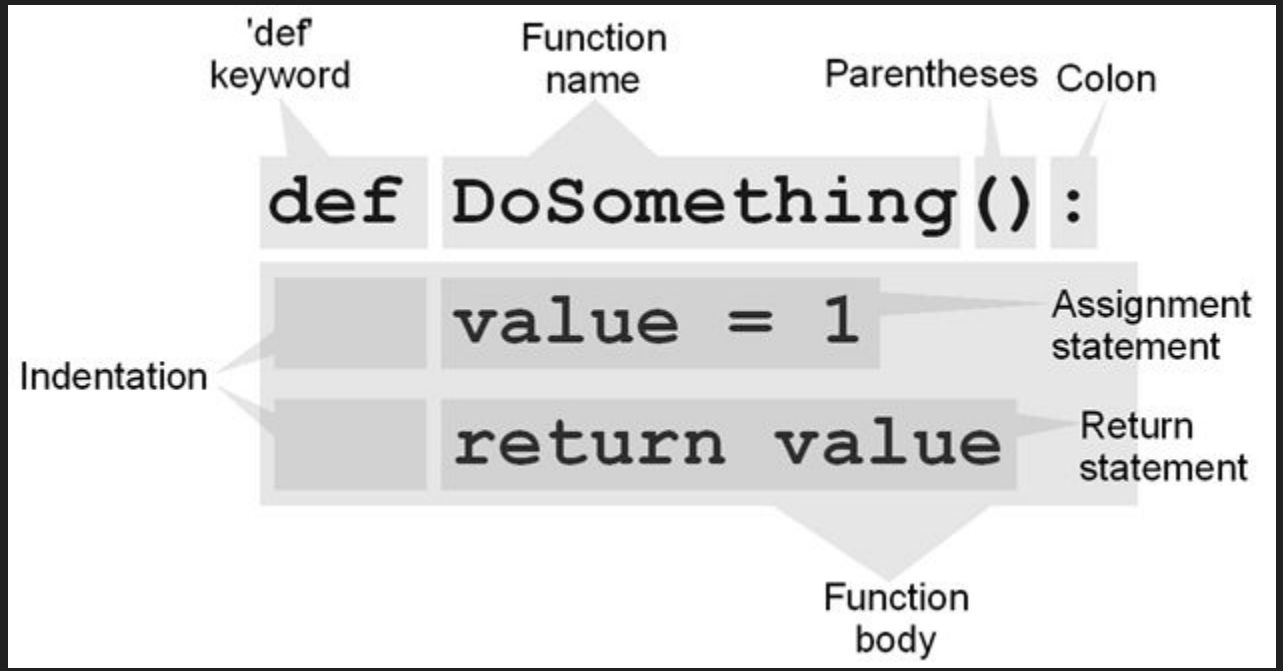
You can think of '*x*' as the generalized parameter, where you can plug in any number you want (the domain, [-inf, inf] in this case).

You can think of the number that the function spits back out as the "return" (the range).

$$f(2) = 2(2)+1$$

$$f(2) = 5$$

Function Syntax



Python Translation

Let's take our function from earlier, $f(x) = 2x+1$, and translate it to Python:

```
def f(x):

    return 2*x + 1
```

Simple, right? Again, '*x*' is the parameter, it can take on the value of anything we pass into it, and the function will spit back out a different value -- the "return" value.

Same concept to functions in mathematics. In programming however, we can do more than just math in our functions.

Why do we need functions?

For the same reason we don’t want to copy-paste chunks of code. Functions offer the ability to ***generalize***. We can keep the same algorithm, but we can call it with different parameters to get different outcomes.

It makes code more readable, easier to debug, and highly reusable -- especially if you make your functions error-proof.

We’ve already seen plenty of functions that are included with Python by default, like `print()` and `input()`, but now we can make our own.

L4-1

Scope

What is scope?

Scope can be defined as *where* the variables you create in your program exist. When you run a program, Python is constantly destroying variables you aren't using in the background to conserve memory on your computer.

Scope becomes extremely relevant when we're talking about functions. Think of functions as existing in a separate world. You can give permission to the function to use certain values (the parameters), and it'll send you values back (the return), but neither you nor the function can access each other's stuff directly without that permission. There is, however, an exception for global scope.

Local Scope

I have a function, and I want the string I initialized inside of it. Why can I not access it?

```
def my_function():
    my_variable = "Hello, there."

print(my_variable)
```

Things that exist in a function *are for the function.* If you want to obtain a value from it, you would have to take the value from a return.

The variable `my_variable` is ***local*** to `my_function`.

Global Scope

This is where things get tricky. Does this code run?

```
x = 10

def my_function():
    print(x)

my_function()
```

`x`, in this case, is considered to be in the *global namespace*.

Global Scope (cont.)

Let’s change up our function a little, does this code run now?

```
x = 10

def my_function():
    x += 1

my_function()
```

Spoiler: it doesn’t. We are trying to change the variable `x`, we end up getting an `UnboundLocalError`, meaning that Python thinks this variable doesn’t exist… so how come we could print it earlier?

Global Scope (cont.)

Variables that are initialized in the *global namespace,* (the space with no indentation), are accessible everywhere in the program, *including the inside of functions*.

However, their value cannot be *changed by a function.* Functions can “look at” these variables, you can copy the value, print it, whatever, as long as you’re not *changing* what the variable holds.

This ***does not*** go the other way around -- you cannot access variables inside functions in the space outside of them.

What if I need to change a global variable inside my function?

Then you need to pass it into the function *as a parameter*, and *overwrite that name with the function's return.* If you need to do this, you likely shouldn't be considering that variable a global variable.

***Important: do not* use the `global` keyword (bad practice for many reasons)**

More on `global` in a second, what does the first thing mean?

`global`

I’ve seen this being used by some people on their projects, and so now is probably a good time to mention to everyone that you’re ***not allowed*** to use it.

There are a multitude of reasons why you shouldn’t use `` `global` ``, but the biggest really comes down to it being bad practice in the industry. I will dock points if you use it. (Item 9 of the coding standard)

I’m not going to go in-depth because it’s not something you should use, but it essentially makes it so that global namespace variables are editable everywhere in the program.