12.9. Global Variables

Variable names that are at the *top-level*, not inside any function definition, are called global.

It is legal for a function to access a global variable. However, this is considered **bad form** by nearly all programmers and should be avoided. This subsection includes some examples that illustrate the potential interactions of global and local variables. These will help you understand exactly how python works. Hopefully, they will also convince you that things can get pretty confusing when you mix local and global variables, and that you really shouldn't do it.

Look at the following, nonsensical variation of the square function.

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

```
1 def badsquare(x):
2     y = x ** power
3     return y
4
5 power = 2
6 result = badsquare(10)
7 print(result)
8
```

```
100
```

Activity: 1 -- ActiveCode (ac11_8_1)

Although the `badsquare` function works, it is silly and poorly written. We have done it here to illustrate an important rule about how variables are looked up in Python. First, Python looks at the variables that are defined as local variables in the function. We call this the **local scope**. If the variable name is not found in the local scope, then Python looks at the global variables, or **global scope**. This is exactly the case illustrated in the code above. `power` is not found locally in `badsquare` but it does exist globally. The appropriate way to write this function would be to pass power as a parameter. For practice, you should rewrite the badsquare example to have a second parameter called power.

There is another variation on this theme of local versus global variables. Assignment statements in the local function cannot change variables defined outside the function. Consider the following codelens example:

Python 3.3

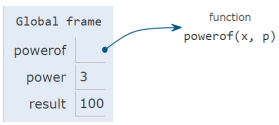
```
1 def powerof(x,p):
2     power = p   # Another dumb mistake
3     y = x ** power
4     return y
5
6 power = 3
7 result = powerof(10,2)
8 print(result)
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed
next line to execute

Visualized using Online Python Tutor by Philip Guo

Frames | Objects



Program output:

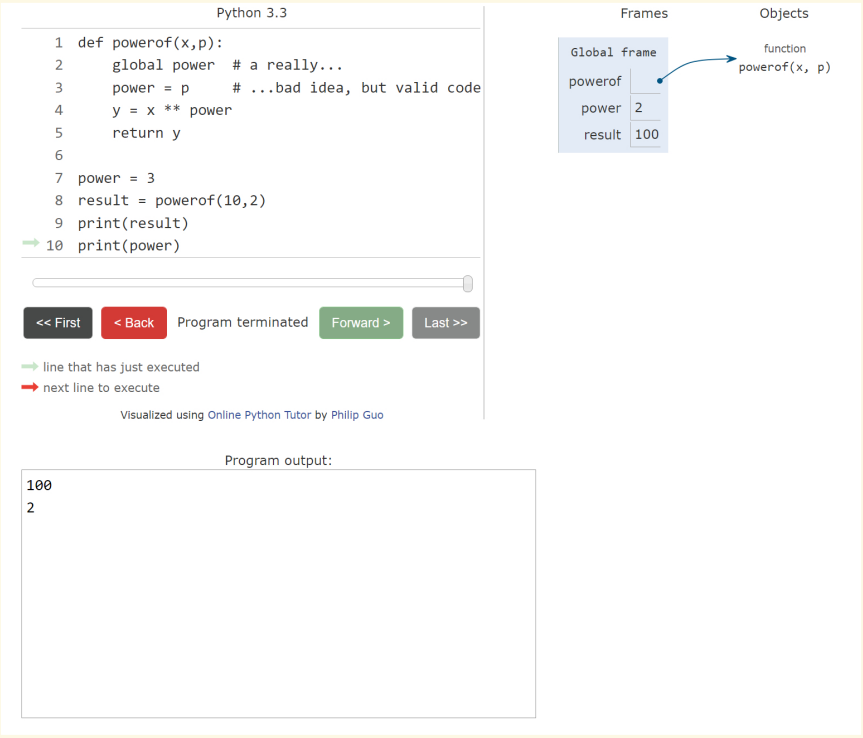
```
100
```

Activity: 2 -- CodeLens: (clens11_8_1)

Now step through the code. What do you notice about the values of variable `power` in the local scope compared to the variable `power` in the global scope?

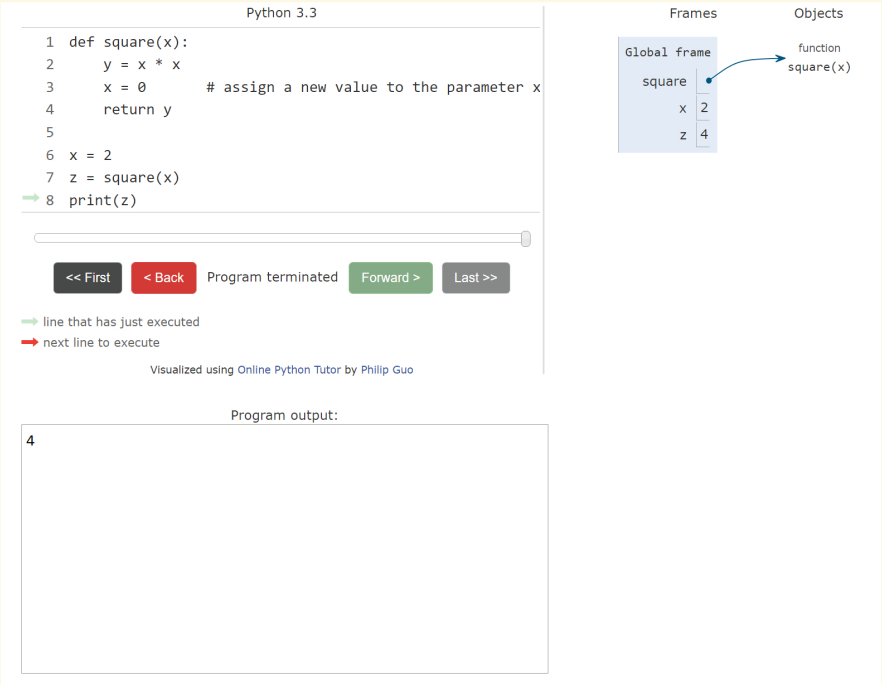
The value of `power` in the local scope was different than the global scope. That is because in this example `power` was used on the left hand side of the assignment statement `power = p`. When a variable name is used on the left hand side of an assignment statement Python creates a local variable. When a local variable has the same name as a global variable we say that the local shadows the global. A **shadow** means that the global variable cannot be accessed by Python because the local variable will be found first. This is another good reason not to use global variables. As you can see, it makes your code confusing and difficult to understand.

If you really want to change the value of a global variable inside a function, you can can do it by explicitly declaring the variable to be global, as in the example below. Again, you should *not* do this in your code. The example is here only to cement your understanding of how python works.



Activity: 3 -- CodeLens: (clens11_8_2)

To cement all of these ideas even further lets look at one final example. Inside the `square` function we are going to make an assignment to the parameter `x` There's no good reason to do this other than to emphasize the fact that the parameter `x` is a local variable. If you step through the example in codelens you will see that although `x` is 0 in the local variables for `square`, the `x` in the global scope remains 2. This is confusing to many beginning programmers who think that an assignment to a formal parameter will cause a change to the value of the variable that was used as the actual parameter, especially when the two share the same name. But this example demonstrates that that is clearly not how Python operates.



Activity: 4 -- CodeLens: (clens11_8_3)

Check your understanding

func-8-1: What is a variable's scope?

- A. Its value
- B. The range of statements in the code where a variable can be accessed.
- C. Its name

Check me Compare me

✔ Correct.

Activity: 5 -- Multiple Choice (question11_8_1)

func-8-2: What is a local variable?

- (●) A. A temporary variable that is only used inside a function
- ( ) B. The same as a parameter
- ( ) C. Another name for any variable

Check me | Compare me

✔ Yes, a local variable is a temporary variable that is only known (only exists) in the function it is defined in.

Activity: 6 -- Multiple Choice (question11_8_2)

func-8-3: Can you use the same name for a local variable as a global variable?

- ( ) A. Yes, and there is no reason not to.
- (●) B. Yes, but it is considered bad form.
- ( ) C. No, it will cause an error.

Check me | Compare me

✔ it is generally considered bad style because of the potential for the programmer to get confused. If you must use global variables (also generally bad form) make sure they have unique names.

Activity: 7 -- Multiple Choice (question11_8_3)

Note

WP: Scope

You may be asking yourself at this point when you should make some object a local variable and when should you make it a global variable. Generally, we do not recommend making variables global. Imagine you are trying to write a program that keeps track of money while purchasing groceries. You may make a variable that represents how much money the person has, called `wallet`. You also want to make a function called `purchase`, which will take the name of the item and its price, and then add the item to a list of groceries, and deduct the price from the amount stored in `wallet`. If you initialize wallet before the function as a variable within the global scope instead of passing it as a third parameter for `purchase`, then an error would occur because wallet would not be found in the local scope. Though there are ways to get around this, as outlined in this page, if your program was supposed to handle groceries for multiple people, then you would need to declare each wallet as a global variable in the functions that want to use wallet, and that would become very confusing and tedious to deal with.

You have attempted 8 of 7 activities on this page

12.8. Variables and parameters are local"> 12.10. Functions can call other functions (Composition)">

12.8. Variables and parameters are local">

12.10. Functions can call other functions (Composition)">Next Section - 12.10. Functions can call other functions (Composition)

| Back to top