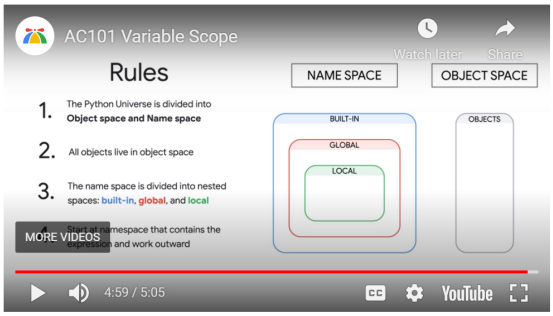
12.8. Variables and parameters are local



Activity: 1 -- Video: (goog_local_vars)

An assignment statement in a function creates a **local variable** for the variable on the left hand side of the assignment operator. It is called local because this variable only exists inside the function and you cannot use it outside. For example, consider again the `square` function:

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

```
1 def square(x):
2     y = x * x
3     return y
4
5 z = square(10)
6 print(y)
7
```

Activity: 2 -- ActiveCode (ac11_7_1)

Error

```
NameError: name 'y' is not defined on line 6
```

Description

A name error almost always means that you have used a variable before it has a value. Often this may be a simple typo, so check the spelling carefully.

To Fix

Check the right hand side of assignment statements and your function calls, this is the most likely place for a NameError to be found.

Try running this in Codelens. When a function is invoked in Codelens, the local scope is separated from global scope by a blue box. Variables in the local scope will be placed in the blue box while global variables will stay in the global frame. If you press the 'last >>' button you will see an error message. When we try to use `y` on line 6 (outside the function) Python looks for a global variable named `y` but does not find one. This results in the error: `Name Error: 'y' is not defined.`

The variable `y` only exists while the function is being executed — we call this its **lifetime**. When the execution of the function terminates (returns), the local variables are destroyed. Codelens helps you visualize this because the local variables disappear after the function returns. Go back and step through the statements paying particular attention to the variables that are created when the function is called. Note when they are subsequently destroyed as the function returns.

Formal parameters are also local and act like local variables. For example, the lifetime of `x` begins when `square` is called, and its lifetime ends when the function completes its execution.

So it is not possible for a function to set some local variable to a value, complete its execution, and then when it is called again next time, recover the local variable. Each call of the function creates new local variables, and their lifetimes expire when the function returns to the caller.

Check Your Understanding

func-7-1: True or False: Local variables can be referenced outside of the function they were defined in.

- A. True
- B. False

Check me | Compare me

✔ Local variables cannot be referenced outside of the function they were defined in.

Activity: 3 -- Multiple Choice (question11_7_1)

func-7-2: Which of the following are local variables? Please, write them in order of what line they are on in the code.

```
numbers = [1, 12, 13, 4]
def foo(bar):
    aug = str(bar) + "street"
    return aug

addresses = []
for item in numbers:
    addresses.append(foo(item))
```

The local variables are

bar aug

Check me Compare me

- Good work!
- Good work!

Activity: 4 -- Fill in the Blank (question11_7_2)

func-7-3: What is the result of the following code?

```
def adding(x):
    y = 3
    z = y + x + x
    return z

def producing(x):
    z = x * y
    return z

print(producing(adding(4)))
```

- A. 33
- B. 12
- C. There is an error in the code.

Check me Compare me

✔ Yes! There is an error because we reference y in the producing function, but it was defined in adding. Because y is a local variable, we can't use it in both functions without initializing it in both. If we initialized y as 3 in both though, the answer would be 33.

Activity: 5 -- Multiple Choice (question11_7_3)

You have attempted 6 of 5 activities on this page

12.7. A function that accumulates">
function that accumulates">
✔ Completed. Well Done!
12.9. Global Variables">
12.9. Global Variables">Next Section - 12.9. Global Variables

| Back to top