



# Unit 3 Lab:

# Variable Scope

# Lesson Objectives

*After this lesson, you will be able to...*

- Define variable scope.
- Use the global keyword to access global variables.
- Explain the order of scope precedence that Python follows when resolving variable names.

# Discussion: Delivering a Letter

What if someone wanted to send Brandi a letter?

If you just had “For Brandi,” the mail carrier would give the letter to the first Brandi they see!

They’d look:

- First in the class. Is there a “Brandi” here? They get the letter!
- No? OK, look in the town. Is there a “Brandi” here? They get the letter!
- No? OK, look in the state. Is there a “Brandi” here? They get the letter!

# Discussion: Your Address

That's why **scope** matters. We might have to get more specific. To correctly deliver the letter, if the mail carrier only looked in the scope of:

Your class:

- You're probably the only Brandi.
- "For Brandi" is fine.

Your town:

- There might be multiple Brandis in the town.
- "For Brandi, on Main Street" is a bit more specific.

In your state:

- There are multiple Main Streets in New York!
- "For Brandi, on Main Street in Brooklyn" is more specific.

## Discussion: What Is **x**?

Python has **scope**, too. We can have many variables with the same name, and Python will look for the most specific one.

In different scopes, you can reuse the same name. Each one is a *completely different* variable.

Functions and classes create individual **local scopes**. A **local variable** doesn't exist outside its local function or class scope.

```
def my_func1():  
    x = 1      # This is a LOCAL variable.  
    print(x)  # 1  
  
def my_func2():  
    x = 5      # This is a DIFFERENT local variable.  
    print(x)  # 5  
  
print(x)  # x is OUT OF SCOPE — no x exists here!
```

# Global Scope

Variables that are in **global scope** can be accessed anywhere. - Python will check for a local variable before using a global one.

```
x = 2

def my_func1():
    x = 1
    print(x) # 1 - Python checks local scopes first.

def my_func2():
    x = 5
    print(x) # 5 - Python checks local scopes first.

my_func1()
```

# Multiple Variables, One Name

Use case: `x` and `y` are frequently used to represent numbers.

Scope is important so they don't interact!

```
def add(x, y):  
    return x + y  
  
def subtract(x, y):  
    return x - y  
  
def multiply(x, y):  
    return x * y  
  
def divide(x, y):  
    return x / y
```

# We Do: Accessing Scopes

Let's start with global scope:

```
foo = 5  
print(foo)  
  
foo = 7  
print(foo)
```



# We Do: Accessing Local Scope

What if we add a variable in a local function scope and try to access it from the global scope?

```
foo = 5

# Delete your other code.
# Add this function and print calls instead.
def coolFunc():
    bar = 8

coolFunc()
print(foo)
print(bar)
```

It fails!

# Scope Can Be Tricky

What do you think happened here?

```
foo = 5
def incrementFoo():
    foo = 6
    print(foo) # prints 6

print(foo) # prints 5
incrementFoo()
print(foo) # prints 5
```

# I Do: The Global Keyword

You can call a global variable intentionally with `global`. \* What do you think happens if you forget the `global` keyword?

```
foo = 5  
  
def incrementFoo():  
    global foo  
    foo += 1  
  
print(foo) # prints 5  
  
incrementFoo()  
  
print(foo) # prints 6!
```

# We Do: Global vs. Local

In the following code, there are three `print` statements. Before you run the code, guess what those `print` statements will print.

run ▶

open in repl.it

main.py history

```
1 # Global variable x
2 x = 1
3
4 # Only local x in this function
5 def my_func1():
6     x = 2 # This is a different, local x
7     print(x) # Print the local x
8
9 # Using global x
10 def mv func2():
```

Python 3.6.1 (default, Dec 2015, 13:05:11)

[GCC 4.8.2] on linux

❏

# You Do: Just a Day in the Jungle

Open a new local file, `piranhas.py`.

- Declare a global variable `piranhas_hungry` and set it to `True`.
- Write two functions, `swing_vine_over_river` and `jump_in_river`.
- In `swing_vine_over_river`, print `Ahhh! Piranhas got me!`.
  - Change `piranhas_hungry` to `False`.
- In `jump_in_river`, if `piranhas_hungry` is `True`, print `I'm not going in there! There are hungry piranhas!`.
  - Otherwise, print `Piranhas are full! Swimming happily through the Amazon!`

```
# Call functions in this order.  
  
jump_in_river()  
  
swing_vine_over_river()  
  
jump_in_river()
```

**Pro tip:** Raise your hand if you need some help!

# We Do: Check Your Answers

- Did you remember the `global` keyword?
- What happens if that keyword is removed?
- Comment out line 4. What happens? Why?

run ▶

open in repl.it

main.py history

```
1  pirhanas_hungry = True
2
3  def swing_vine_over_river():
4      # global pirhanas_hungry
5      print("Ahhhh! Pirhanas got me!")
6      pirhanas_hungry = False
7
8  def jump_in_river():
9      global pirhanas_hungry
10     if not pirhanas_hungry:
```

Python 3.6.1 (default, Dec 2015, 13:05:11)  
[GCC 4.8.2] on linux  
✖

# Summary and Q&A

Python checks **scope** to find the right variable.

- Functions and classes create individual **local scopes**.
  - A `local` variable doesn't exist outside its local function or class scope.
- Any variable declared or assigned outside of any function or class is considered “global.”
  - Variables that are in **global scope** can be accessed anywhere.

Python will check for a `local` variable before using a `global` one.

There can be more levels. Python always works from the inside out — keep that in mind as your programs get more advanced!

# Additional Resources

- [Global vs. Local Variables](#)
- [Variables and Scope](#)
- [Nested Functions — What Are They Good For?](#)