Local vs. Global Variables

Local variables

Local variables exist only within a function’s body. They cannot be referred outside of it.

Parameters are also local variables that are assigned a value when the function is invoked. They cannot be referred outside the function too.

Local variables in the Frame Model

We’ve seen numerous examples of functions that use local variables, but we haven’t explained how local variables work in the execution model with function frames.

We’ll do that now with the `hypotenuse2` function:

```
def hypotenuse2(a, b):
    sqa = square(a)
    sqb = square(b)
    sqsum = sqa + sqb
    return math.sqrt(sqsum)
```
Functions w/local variables: hypotenuse2

def hypotenuse2(a, b):
    sqa = square(a)
    sqb = square(b)
    sqsum = sqa + sqb
    return math.sqrt(sqsum)

hypotenuse2(3, 4)

a 3 b 4
sqa = square(a)
sqb = square(b)
sqsum = sqa + sqb
return math.sqrt(sqsum)

square frame

Local Vars 5

Functions w/local variables: hypotenuse2

a 3 b 4
sqa = square(3)
sqb = square(b)
sqsum = sqa + sqb
return math.sqrt(sqsum)

sqa = 9
sqb = square(4)
sqsum = sqa + sqb
return math.sqrt(sqsum)

square frame

Local Vars 6

Functions w/local variables: hypotenuse2

a 3 b 4
sqa = 9
sqb = square(b)
sqsum = sqa + sqb
return math.sqrt(sqsum)

sqa = 9
sqb = 16
sqsum = 9 + sqb
return math.sqrt(sqsum)

square frame

Local Vars 7

Functions w/local variables: hypotenuse2

a 3 b 4
sqa = square(9)
sqb = square(16)
sqsum = sqa + sqb
return math.sqrt(sqsum)

sqa = 9
sqb = 16
sqsum = 25
return math.sqrt(sqsum)

square frame

Local Vars 8

Functions w/local variables: hypotenuse2

a 3 b 4
sqa = square(9)
sqb = square(16)
sqsum = sqa + sqb
return math.sqrt(sqsum)

sqa = 9
sqb = 16
sqsum = 25
return math.sqrt(sqsum)

return 5.0
Global Variables / Global vs Local Scope

Variables assigned outside any function are known as **global variables**. If a variable is assigned within a function definition, it is assumed to be a **local variable** of the function unless the **global** declaration is used to indicate it references a global variable outside the function instead.

```python
x1 = 5
print(x1)

def f1():
    x1 = 7
    print(x1)

f1()
print(x1)
```

```
5
7
```

Another kind of a side effect: changing a global variable.

```python
x2 = 5
print(x2)

def f2():
    global x2
    x2 = 7
    print(x2)

def f3():
    x3 = 7
    print(x3)

def f4():
    global x4
    x4 = 7
print(x4)
```

```
5
7
7
7
```

Counter function

How can we define a zero-parameter count function that returns the number of times it has been called?

```python
c = 0 # global variable storing the current count

def count():
    global c
    # What happens if we forget this?
    c = c + 1
    return c
```

```
count() → 1
count() → 2
count() → 3
count() → 4
```

Python Tutor with globals and locals

Python Tutor: [http://www.pythontutor.com/visualize.html](http://www.pythontutor.com/visualize.html)

---

Can we change a value in global variable without the **global** keyword?

Yes – if the value is mutable!

```python
from cs1graphics import *
circleCounter = Circle(1) # circles must have positive radius
def count():
    circleCounter.setRadius(1 + circleCounter.getRadius())
    return circleCounter.getRadius()
```

```
count() → 2.0 # radius is a float
count() → 3.0
count() → 4.0
```

Note: the value in **circleCounter** stays the same **Circle** object, but the state of that object changes over time.