LOCAL

Lesson 1: functions don't execute unless they're called

def set\_x():

x = 100

print(x)

Lesson 2: variables created in a function die after function returns

def set\_x():

x = 100

set\_x()

print(x)

Lesson 3: variables start fresh every time a function is called again

def count():

x = 1

x += 1

print(x)

count()

count()

count()

Lesson 4: you can't see the variables of other function invocations, even those that call you

def display\_x():

print(x)

def main():

x = 100

display\_x()

main()

GLOBAL

Lesson 5: you can generally just **use** global variables inside a function

msg = 'hello' # global, outside any func

def greeting():

print(msg)

print('before: ' + msg)

greeting()

print('after: ' + msg)

Lesson 6: if you do an assignment to a variable in a function, Python assumes you want it local

msg = 'hello'

def greeting():

msg = 'welcome!'

print('greeting: ' + msg)

print('before: ' + msg)

greeting()

print('after: ' + msg)

Lesson 7: assignment to a variable should be before its use in a function, even if there's

a global variable with the same name

msg = 'hello'

def greeting():

print('greeting: ' + msg)

msg = 'welcome!'

print('before: ' + msg)

greeting()

print('after: ' + msg)

Lesson 8: use a global declaration to prevent Python from creating a  
 local variable when you want a global variable

msg = 'hello'

def greeting():

global msg

print('greeting: ' + msg)

msg = 'welcome!'

print('before: ' + msg)

greeting()

print('after: ' + msg)

ARGUMENT PASSING

Lesson 9: in Python, arguments are "passed by value", meaning

reassignments to a parameter don't change the argument outside

def f(x):

x = 'B'

print('inside: ' + x)

val = 'A'

print('before: ' + val)

f(val)

print('after: ' + val)

Lesson 10: it's irrelevant whether the argument (outside) and  
 parameter (inside) have the same variable name

x = 'A'

def f(x):

x = 'B'

print('inside: ' + x)

print('before: ' + x)

f(x)

print('after: ' + x)