Learning Quiz 23: Global versus Local Namespaces

Due Nov 13 at 1pm **Points** 5 **Questions** 5 **Available** until Dec 4 at 11:59pm **Time Limit** None
Allowed Attempts Unlimited

Instructions

Prior to completing this quiz, be sure to read:

- Section 7.2: Global versus Local Namespaces (p. 211-215)

Please also go over Practice Problem 7.2 in the textbook (solution at the end of the chapter) before attempting this quiz.

This quiz was created for learning purposes. You may attempt this quiz as many times as you would like. The highest score *prior to the deadline* will count towards the final course grade. No late submissions will be accepted.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	7 minutes	4 out of 5

Score for this attempt: **4** out of 5
Submitted Nov 9 at 6:40pm
This attempt took 7 minutes.

Question 1

1 / 1 pts

We have seen that a variable defined within a function, even with the same variable name, will not change the variable defined outside of the function. Notice that outside the function, x remains the value 4 at the end of the function:

```
x = 4
def myfun():
    x = 5
    print(x)
myfun()
print(x)
```

However, what if the goal of our function was to change the value of our global variable x? We will need to use the global reserved word to indicate that we are referencing the global variable.

```
x = 4
def myfun():
    global x
    x = 5
    print(x)
myfun()
print(x)
```

Consider the following program.

```
x = 4
y = 10
def myfun():
    global y
    x = 5
    y = 21
    print(x, y, end = " ")
myfun()
print(x, y)
```

What prints?

- 5 21 5 10
- 5 10 4 21
- 5 21 5 21
- 5 21 4 10
- Correct! 5 21 4 21

Question 2

1 / 1 pts

Like variables, we can define functions within functions and those local functions can only be accessed locally within the function. Note that inner functions (like innerFun() below) are masked from the larger Python shell. In the following example, note that we can call innerFun() within bigFun(). However, we cannot call innerFun() in our Python shell:

```
def bigFun():
    print('We are in bigFun()')
    def innerFun():
        print('We are in innerFun()')
    print('We have created innerFun() but we have not called it')
    innerFun()
    print('We just called innerFun()')

print('Lets call bigFun()')
bigFun()

print('This next line will give us an error')
innerFun() ## because innerFun() can only be called inside bigFun()
```

Consider the following code:

```
def yourFun():
    pass

def herFun():
    pass

def myFun():
    def yourFun():
        print('Hello')
        def hisFun():
            print("oops")
    def noFun():
        pass
```

```
    print('Bye')

##### Call function here #####
print('End')
```

Select all of the functions that can be called on at the indicated line.

- [ ] noFun()
- [x] herFun() — Correct!
- [x] yourFun() — Correct!
- [ ] hisFun()
- [x] myFun() — Correct!

Question 3

1 / 1 pts

You can continue to create new same-named variables within nested functions. However, the x value will only remain for the scope of the function:

```
x = 1
print("Outside functions x is",x)
def outFun():
    x = 2
    print("outFun() x is", x)
    def inFun():
        x = 3
        print("inFun() x is",x)
    inFun()
    print("outFun() x is STILL", x)
outFun()
print("global x is STILL", x)
```

You can access global x within a function, and create a local x within the inner function:

```
x = 1
print("Outside functions x is",x)
def outFun():
    global x
    x = 2
    print("outFun() x is", x)
    def inFun():
        x = 3
        print("inFun() x is",x)
    inFun()
    print("outFun() x is STILL", x)
outFun()
print("global x is NOW", x)
```

You can similarly access a global x within an inner function, but create a local x for the outer function:

```
x = 1
print("Outside functions x is",x)
def outFun():
    x = 2
    print("outFun() x is", x)
    def inFun():
        global x
        x = 3
        print("inFun() x is",x)
    inFun()
    print("outFun() x is STILL", x)
```

```
outFun()
print("global x is NOW", x)
```

Of course, this can all get very confusing, so it's best to use descriptive and distinct variable names. Be sure to follow the example code by hand to keep track of which x we're modifying.

Consider the following code:

```
x = 1
def outFun():
    global x
    x = 2
    def inFun():
        global x
        x = 3
    inFun()
outFun()
```

What is the value of x by the end of the code?

- [ ] None
- [ ] 2
- [x] 3

Correct!

- [ ] 1

Question 4

1 / 1 pts

To access the outer function x and not the global x, use the reserved word **nonlocal**:

```
x = 1
print("Outside functions x is",x)
def outFun():
    x = 2
    print("outFun() x is", x)
    def inFun():
        nonlocal x
        x = 3
        print("inFun() x is",x)
    inFun()
    print("outFun() x is NOW", x)
outFun()
print("global x is STILL", x)
```

Consider the following code:

```
x = 1
print("Outside functions x is",x)
def outFun():
    global x
    x = 2
    print("outFun() x is", x)
    def inFun():
        nonlocal x
        x = 3
        print("inFun() x is",x)
    inFun()
    print("outFun() x is NOW", x)
outFun()
print("global x is STILL", x)
```

What happens when we run the code?

- [ ] inFun() will modify the global x value to 3
- [x] Error, nonlocal x does not exist

Correct!

- [ ] The program will run and ignore the nonlocal x line since outFun() references global x
- [ ] outFun() will modify the global x value to 2, and inFun will modify the outFun() x value to 3

Question 5

0 / 1 pts

Even with different variable names, the scope of variables are limited to the level where they were created. Consider the following program:

```
x = 1
def outFun(y):
    z = 3
    def inFun(v):
        w = 5
    inFun(2)
outFun(4)

##### print variables here #####
```

Which variables can be printed at the indicated line? Select all.

Correct!

- [x] x
- [ ] w

You Answered

- [x] z
- [ ] v

You Answered

- [x] y

Quiz Score: **4** out of 5