**1. What is the result of the code, and explain?**

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

**OUTPUT:**

iNeuron

As the func function will be invoked, print(X) will run. It will try to find X inside the function first, as

we do not have X inside the function, it will use the outer definition for X and print the value.

**2. What is the result of the code, and explain?**

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

**OUTPUT:**

iNeuron

Due to the local scope, the value of X did not change. Hence iNeuron was printed.

**3. What does this code print, and why?**

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

**OUTPUT:**

NI

iNeuron

Both the values were printed as the value of X is different in global and local scopes.

**4. What output does this code produce? Why?**

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

**OUTPUT:**

Defining X as global inside the function will let us updated the value of X from inside the function.

**5. What about this code—what’s the output, and why?**

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

def nested():

print(X)

nested()

>>> func()

>>> X

**OUTPUT:**

NI

iNeuron

The nested function will consider the local scope and print the value ‘NI’.

**6. How about this code: what is its output in Python 3, and explain?**

>>> def func():

X = 'NI'

def nested():

nonlocal X

X = 'Spam'

nested()

print(X)

>>> func()

OUTPUT:

Spam

nonlocal X indicates that the we can change the global value of X. Hence the output is ‘Spam’.