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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

**ANS**

iNeuron because X is defined outside and can be used as a public constant

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

**ANS**

iNeuron

X inside a func() is a function variable and altering it doesn’t affect X outside the function

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

**ANS**

iNeuron

X inside a func() is a local variable and altering it doesn’t affect X outside the function

4. What output does this code produce? Why?

Print inside

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

**ANS**

NI

Since X is made as global changes will be reflected throughout

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

def nested():

print(X)

nested()

>>> func()

>>> X

**ANS**

iNeuron

Changes to X inside func() will be only valid inside the scope of that function to make it change we should give it as global

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()

**ANS**

Spam

Changes to X inside func() will be only valid inside the scope of that function to make it change will not be reflected But inside nested the variable is declared as nonlocal so by changing it it will be reflected across the code.