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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

ANSWER.

iNeuron

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

ANSWER.

iNeuron

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

ANSWER.

NI

iNeuron

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

ANSWER.

NI

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

def nested():

print(X)

nested()

>>> func()

>>> X

ANSWER.

NI

'iNeuron'

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()

ANSWER.

In this code, the `func()` function defines a local variable `X` with the value `'NI'`. Within `func()`, there's a nested function `nested()` that attempts to modify the value of `X` using the `nonlocal` keyword. The `nonlocal` keyword is used to indicate that `X` refers to the nearest enclosing scope (which is `func()` in this case) rather than creating a new local variable.

However, in Python 3, the `nonlocal` keyword cannot be used to modify variables from an outer scope if they are not defined in an enclosing scope. Since `X` is not defined in an outer scope of `nested()`, using `nonlocal X` will result in a `SyntaxError`.

So, when the code is executed in Python 3, it will raise a `SyntaxError` at the line `nonlocal X`.