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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

ANS :- iNeuron

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

ANS :- iNeuron

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

ANS :- NI

iNeuron

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

ANS :-

The code snippet provided defines a global variable `X` with the value `'iNeuron'`. It also defines a function `func()` that uses the `global` keyword to indicate that the variable `X` inside the function should refer to the global variable `X` defined outside the function. Within the function, it assigns the value `'NI'` to the global variable `X`.

After calling the function `func()` and modifying the global variable `X, the code prints the value of the global variable `X` using `print(X)`.

The expected output would be:

```

NI

```

The reason for this output is that by using the `global` keyword inside the function `func()`, the assignment `X = 'NI'` modifies the global variable `X` instead of creating a new local variable. Therefore, when the value of `X` is printed outside the function, it reflects the modified value `'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

ANS :- The output of this code would be `'iNeuron'`.

In this code, the global variable `X` is assigned the value `'iNeuron'`. The function `func()` defines a local variable `X` with the value `'NI'`. Inside `func()`, there is a nested function `nested()` that prints the value of `X` within its scope.

When `func()` is called, it executes the nested function `nested()` which prints the value of the local variable `X` within its scope. Since `nested()` is called within the scope of `func()`, it can access the local variable `X` defined in `func()` and prints `'NI'`.

However, when `X` is printed outside the function, it refers to the global variable `X` defined outside the function `func()`. The global variable `X` was not modified within the function, so its value remains `'iNeuron'`.

Therefore, the output will be `'iNeuron'` when `X` is printed outside the function.

Output:

```

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()

ANS :- The code will raise a `SyntaxError` in Python 3.

The reason for the `SyntaxError` is that the `nonlocal` keyword is used within the function `nested()` to indicate that the variable `X` is a nonlocal variable, meaning it is a variable defined in an outer enclosing scope (in this case, within the scope of the function `func()`).

However, there is no variable `X` defined in any outer enclosing scope of the `nested()` function. Therefore, the `nonlocal` keyword cannot be used for `X`, and Python raises a `SyntaxError` in this case.

To fix this error, the code needs to have an intermediate scope where `X` is defined and accessed by both `nested()` and `func()`.