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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

The code defines a global variable X with the value 'iNeuron', and then it defines a function func that prints the value of X. When you call func(), it prints the value of the global variable X, which is 'iNeuron'. The result is that it will print "iNeuron".

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

In this code, X is defined as a global variable with the value 'iNeuron'. The function func defines a local variable X with the value 'NI!', but this variable is local to the function and does not affect the global X. When you call func(), it sets the local X to 'NI!', but the global X remains unchanged. After calling func(), when you print X, it will still print "iNeuron". The result is that it will print "iNeuron".

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

Similar to the previous code, the global variable X is defined as 'iNeuron'. The function func defines a local variable X with the value 'NI', but this local variable is separate from the global X. When you call func(), it sets the local X to 'NI' and prints it within the function. However, the global X remains unchanged. After calling func(), when you print X, it will still print "iNeuron". The result is that it will print "NI" (inside the function) and then "iNeuron" (outside the function).

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

In this code, the function func uses the global keyword to indicate that X inside the function should refer to the global variable X. When you call func(), it sets the global X to 'NI', and then when you print X, it will print "NI" because the global X has been modified by 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

This code defines a global variable X with the value 'iNeuron', and it defines a function func that sets a local variable X with the value 'NI'. Inside the func function, there's another nested function called nested, which attempts to print X. However, nested does not have its own X variable, so it looks for X in the nearest enclosing scope, which is the func function. It finds the local variable X set by func, so it prints "NI". When you call func(), it runs the nested function, and then you print X, which is the global variable and still has the value "iNeuron". The result is that it will print "NI" and then "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()

In this code, the func function defines a local variable X with the value 'NI'. The nested function nested uses the nonlocal keyword to indicate that X should refer to the nearest enclosing scope that is not global, which is the func function. It sets X to 'Spam' inside the func function. When you call func(), it modifies the X variable in the nearest enclosing scope, which is func, so it sets X to "Spam". When you print X within the func function, it will print "Spam". The result is that it will print "Spam".