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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

**Ans:-** The code will output the value “ineuron” . Here’s an explanation:-

The variable ‘**x’** is assigned the string value “**ineuron”**  in the global scope using “**x = ineuron”.**

The “**func()”**  is defined, which attempts tom print the value of the variable ‘**x’.** When the “**func()”**

function is called with “**func()”** , it accesses the value of x from the global scope, since the variable is

not defined within the function. The ‘**print()’** statement inside the “**func()**” prints the value of x,

which is “**ineuron”.** As a result, when the code is executed, it will output ‘**ineuron**’.

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

**Ans:-** The code will output 'iNeuron'. Here's an explanation:

The variable X is assigned the string value 'iNeuron' in the global scope using the statement X = 'iNeuron'.

The function func() is defined, which declares a new local variable X and assigns it the value 'NI!'. This local variable X is distinct and separate from the global variable X.

When the func() function is called with func(), it assigns the value 'NI!' to the local variable X. However, since this assignment is within the scope of the function, it does not affect the global variable X.

After the func() function call, the statement print(X) is executed in the global scope. Since the local variable X inside the function does not affect the global variable, the value of the global X remains unchanged, which is 'iNeuron'.

As a result, when the code is executed, it will output 'iNeuron'.

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

**Ans:-** The code will print:- NI

iNeuron

Here's an explanation:

The variable X is assigned the string value 'iNeuron' in the global scope using the statement X = 'iNeuron'.

The function func() is defined, which declares a new local variable X and assigns it the value 'NI'. This local variable X is distinct and separate from the global variable X.

When the func() function is called with func(), it assigns the value 'NI' to the local variable X. The print(X) statement inside the function then prints the value of the local X, which is 'NI'.

After the func() function call, the statement print(X) is executed in the global scope. It prints the value of the global variable X, which remains unchanged as 'iNeuron'.

As a result, when the code is executed, it first prints 'NI' inside the func() function, and then it prints 'iNeuron' in the global scope.

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

**Ans:-** The code will print :- NI

Here's an explanation:

The variable X is assigned the string value 'iNeuron' in the global scope using the statement X = 'iNeuron'.

The function func() is defined, but it does not contain any code.

After defining the function, the global variable X is redefined within the global scope with the value 'NI' using the statement X = 'NI'.

When the func() function is called with func(), it does not perform any action.

After the func() function call, the statement print(X) is executed in the global scope. It prints the value of the global variable X, which has been reassigned to 'NI' in step 3.

As a result, when the code is executed, it prints '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 code will print :- **ineuron.**

Here's an explanation:

The variable X is assigned the string value 'iNeuron' in the global scope using the statement X = 'iNeuron'.

The function func() is defined, which declares a new local variable X and assigns it the value 'NI'. This local variable X is distinct and separate from the global variable X.

The function nested() is defined, which contains a print(X) statement. It tries to access the value of X from the surrounding scope (the global scope and the func() function's scope).

When the func() function is called with func(), it declares and assigns the value 'NI' to the local variable X. However, since the nested() function is not called within the func() function, the local variable X within func() does not affect the scope of the nested() function.

After the func() function call, the statement print(X) is executed in the global scope. It prints the value of the global variable X, which is 'iNeuron' assigned in step 1.

As a result, when the code is executed, it prints '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 wil print:- **Spam**

Here's an explanation:

The function func() is defined, which contains another function nested().

Inside the nested() function, the nonlocal keyword is used to indicate that the variable X is nonlocal, meaning it refers to the variable X in the nearest enclosing scope, which is the scope of func().

The variable X within nested() is never assigned a value or used in any other way, so it does not have any effect on the execution or output of the code.

After defining the nested() function, the variable X within the scope of func() is assigned the value 'Spam'.

The nested() function is then called, but since it does not perform any actions, it does not affect the value of X.

Finally, the statement print(X) is executed in the scope of func(), which prints the value of X, which is 'Spam'.