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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

Sol:

iNeuron

iNeuron

iNeuron

iNeuron

iNeuron

iNeuron

the function prints the value INeuron as x is theglobal variable defined outside the function.

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

Sol.

iNeuron

this code prints the value iNeuron as X=iNeuron acts as the variable inside the func makesit local and hides the global of the same name .the printstatement finds the variable unchanged.

..

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

Sol.

iNeuron.

This code also prints the value iNeuron as the referencein the print statementfinds the global variable.

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

Sol.

NI.

. This time it just prints 'NI' because the global declaration forces the variable assigned inside the function to refer to the variable in the enclosing global scope.

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

def nested():

print(X)

nested()

>>> func()

>>> X

Sol.

'iNeuron’

The output in this case is again 'iNeuron', because the print statement in the nested function finds the name in the enclosing function’s local scope, and the print at the end finds the variable in the global scope.

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()

Sol.

**SyntaxError:** no binding for nonlocal 'X' found

nonlocal variable is not used for a variable which is not in nesting scope.