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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

**Output: iNeuron**

**The func() would call this function and execute the code written inside this function body. The body of this function has a print statement which prints the arguments on console. X has a string value ‘iNeuron’ which is printed console.**

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

**Output:**

**iNeuron**

**The function called has assignment statement that assigns a local value to variable X which contains ‘NI’. This value will remain in existence till the time the scope of this function doesn’t get over. Once the execution gets over and control comes outside of the function, the global value of X is printed.**

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

**Output:**

**NI**

**iNeuron**

**In case of conflict, function considers the local value of X as the only value. Hence, the print statement within the scope of function will print only local value of X. Once the function finishes its execution, the scope of local X value vanishes and the only value available is ‘iNeuron’.**

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

**Output: NI**

**By default, all variables defined in a function are local. Global keyword is used to define a variable as global in local context. After defining a variable as global, the interpreter will change the global value of X and that same change is reflected outside of that function too.**

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

def nested():

print(X)

nested()

>>> func()

>>> X

**Output:**

**NI**

**‘iNeuron’**

**The above code shows an example of enclosing scope in which we have a function within another function. When inner function tries to access a variable, it first checks out its local scope first. If that variable is not present in its local scope, then it searches it in its enclosing scope i.e. the scope where the inner function is defined. In the above code, nested() doesn’t have its local variable X so it went straight for the enclosing scope and found X=’NI’ and printed the same.**

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()

**Output: Spam**

**With keyword nonlocal we are defining scope of X as nonlocal. By doing so, we will be able to access the nonlocal X and can make changes within the nested function too. This leads to change in value of X from ‘NI’ to ‘Spam’.**