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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

Ans :   
The code you provided defines a variable X with the value 'iNeuron' and then defines a function func(). The function func() attempts to print the value of the variable X.

When the **func()** function is called using **func()**, it will try to print the value of **X**. However, since **X** is defined outside the function and is not passed as an argument or defined within the function, the function will look for **X** in its local scope and then in its global scope.

In this case, since **X** is not defined within the function and it is not a global variable, the function will raise a **NameError** and fail to execute. The code will result in a **NameError** with the message **name 'X' is not defined**.

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

Ans : The code you provided defines a variable X with the value 'iNeuron' and then defines a function func(). Inside the func() function, a local variable X is defined with the value 'NI!'.

When the **func()** function is called using **func()**, it creates a local variable **X** within the function scope and assigns it the value **'NI!'**. This local variable **X** shadows the global variable **X** with the value **'iNeuron'**.

After the function call, the **print(X)** statement is executed. It attempts to print the value of the global variable **X**, which is still **'iNeuron'**. Since the local variable **X** defined within the function does not affect the value of the global variable **X**, the code will output **'iNeuron'**.

Therefore, the result of the code will be:

Copy code

iNeuron

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

Ans : code provided defines a variable X with the value 'iNeuron' and then defines a function func(). Inside the func() function, a local variable X is defined with the value 'NI', and it is printed using the print() function.

When the **func()** function is called using **func()**, it creates a local variable **X** within the function scope and assigns it the value **'NI'**. The **print(X)** statement inside the function will print the value of the local variable **X**, which is **'NI'**.

After the function call, the **print(X)** statement is executed outside the function. It attempts to print the value of the global variable **X**, which is still **'iNeuron'**. The local variable **X** defined within the function does not affect the value of the global variable **X**.

Therefore, the code will print:

NI iNeuron

The first **print(X)** statement inside the **func()** function prints **'NI'**, which is the value of the local variable **X** within the function scope. The second **print(X)** statement outside the function prints **'iNeuron'**, which is the value of the global variable **X**

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

Ans : code provided defines a global variable X with the initial value 'iNeuron' and then defines a function func(). Inside the func() function, it uses the global keyword to indicate that the variable X refers to the global scope, and it assigns the value 'NI' to the global variable X.

When the **func()** function is called using **func()**, it modifies the global variable **X** to have the value **'NI'**.

After the function call, the **print(X)** statement is executed. It attempts to print the value of the global variable **X**, which is now **'NI'** due to the modification made by the **func()** function.

Therefore, the code will print:

NI

The **print(X)** statement will output **'NI'**, which is the value of the modified global variable **X** within the **func()** function. The **global** keyword ensures that the assignment inside the function affects the global scope variable **X**, and that modified value is printed outside 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

Ans : The code you provided defines a global variable X with the initial value 'iNeuron' and then defines a function func(). Inside the func() function, there is another nested function called nested(), which attempts to print the value of the variable X.

When the **func()** function is called using **func()**, it creates a local variable **X** within the function scope with the value **'NI'**.

Inside the **nested()** function, it tries to print the value of the variable **X**. However, since there is no local variable **X** defined within the **nested()** function, it will search for **X** in the enclosing scope, which is the scope of the **func()** function. It finds the local variable **X** defined in the **func()** function and prints its value, which is **'NI'**.

After the function call, the **print(X)** statement is executed outside the function. It attempts to print the value of the global variable **X**, which is still **'iNeuron'**. The local variable **X** defined within the **func()** function does not affect the value of the global variable **X**.

Therefore, the code will print:

NI 'iNeuron'

The **print(X)** statement inside the **nested()** function will output **'NI'**, which is the value of the local variable **X** within the **func()** function scope. The **print(X)** statement outside the function will output **'iNeuron'**, which is the value of the global variable **X**.

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 : code provided defines a function func(), inside which there is another nested function called nested(). The nested() function uses the nonlocal keyword to indicate that the variable X refers to the nearest enclosing scope, which is the scope of the func() function.

When the **func()** function is called using **func()**, it creates a local variable **X** within the function scope with the value **'NI'**.

Inside the **nested()** function, the **nonlocal** keyword is used to indicate that the variable **X** is not local to the **nested()** function but belongs to the enclosing **func()** function's scope. The statement **X = 'Spam'** modifies the value of **X** in the **func()** function's scope.

After modifying the value of **X**, the **nested()** function completes its execution and control returns to the **func()** function. The **print(X)** statement is then executed within the **func()** function, which prints the modified value of **X**, which is **'Spam'**.

Therefore, the output of the code will be:

Copy code

Spam

The **print(X)** statement inside the **func()** function will output **'Spam'**, as the value of **X** was modified by the **nested()** function using the **nonlocal** keyword.