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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

Sol:-

The code will print the string 'iNeuron' when the function `func()` is called.

The variable `X` is defined outside of the function `func()`, which means it has a global scope and is

accessible within the function. When the function `func()` is called, it prints the value of `X`, which is

'iNeuron'.

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

Sol:-

The output will be 'iNeuron'.

In the given code, the variable X is defined with the value 'iNeuron' outside the function. Then, inside

the function, a new local variable X is defined and assigned a new value 'NI!'. However, this local

variable X is not used anywhere, so it does not change the value of the global variable X. Therefore,

when we print the value of X outside the function, it remains 'iNeuron'.

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

Sol:-

NI

iNeuron

In the first line, we define a global variable `X` with the value 'iNeuron'. Then we define a function

`func` which defines a local variable `X` with the value 'NI'. When we call the function `func` with

`func()`, it prints the value of `X` within the function, which is 'NI'. However, when we print the value

of `X` outside the function, it still holds the value 'iNeuron' because it is a global variable and not

affected by the local variable with the same name within the function.

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

Sol:-

The output will be 'NI'.

In this code, the global keyword is used to declare the variable 'X' as a global variable inside the

function 'func()'. This means that any changes made to this variable inside the function will affect the

global variable 'X' declared outside the function.

So, when the function 'func()' is called, it sets the value of global variable 'X' to 'NI'. When we print

the value of 'X' outside the function, it prints the updated value 'NI' instead of the initial value

'iNeuron'.

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:-

The code will output `NI` followed by `iNeuron`.

The `func` function defines a local variable `X` and sets its value to `'NI'`. It then defines another

function `nested`, which is called within `func`. The `nested` function does not define a variable `X`,

so it looks for the value of `X` in the enclosing scope, which is the `func` function. Since `X` is defined

in `func`, `nested` prints the value `'NI'`.

After calling `func`, the value of `X` is still `'iNeuron'`, since the `X` defined in `func` is a local variable

and does not affect 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()

Sol:-

The output of the code will be "Spam".

The function `func()` defines a local variable `X` and a nested function `nested()` that modifies the

value of `X` using the `nonlocal` keyword. When `func()` is called, it executes `nested()` which

changes the value of `X` to 'Spam', and then prints it. Finally, the `print` statement in `func()` also

prints the value of `X`, which has been changed to 'Spam' by the `nested()` function.