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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

def nested():

print(X)

nested()

>>> func()

>>> 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()

ANSWER

1. \*\*Code 1:\*\*

```python

X = 'iNeuron'

def func():

print(X)

func()

```

Output: `iNeuron`

Explanation: The function `func` does not have its own local variable named `X`, so it looks for a variable with that name in the global scope and prints its value, which is 'iNeuron'.

2. \*\*Code 2:\*\*

```python

X = 'iNeuron'

def func():

X = 'NI!'

func()

print(X)

```

Output: `iNeuron`

Explanation: The `X` variable within the `func` function is a local variable, and it is different from the global `X` variable. Changing the local `X` does not affect the global `X`.

3. \*\*Code 3:\*\*

```python

X = 'iNeuron'

def func():

X = 'NI'

print(X)

func()

print(X)

```

Output:

```

NI

iNeuron

```

Explanation: In the `func` function, a local variable `X` is created, and it is printed within the function. This local `X` does not affect the global `X`, which is printed outside the function.

4. \*\*Code 4:\*\*

```python

X = 'iNeuron'

def func():

global X

X = 'NI'

func()

print(X)

```

Output: `NI`

Explanation: The `global` keyword inside the `func` function indicates that the function should modify the global variable `X`. Therefore, the global `X` is changed to 'NI', and this change is reflected when printing the global `X`.

5. \*\*Code 5:\*\*

```python

X = 'iNeuron'

def func():

X = 'NI'

def nested():

print(X)

nested()

func()

print(X)

```

Output:

```

NI

iNeuron

```

Explanation: The `nested` function first looks for the variable `X` in the local scope of `nested`. Since it doesn't find it there, it then looks in the enclosing scope (the `func` function) and finds `X`. The `X` in the global scope remains unchanged.

6. \*\*Code 6:\*\*

```python

def func():

X = 'NI'

def nested():

nonlocal X

X = 'Spam'

nested()

print(X)

func()

```

Output: `Spam`

Explanation: The `nonlocal` keyword inside the `nested` function indicates that it should modify the nearest enclosing variable named `X`, which is in the `func` function. So, `X` is changed to 'Spam' within the `func` function, and this change is reflected when printing `X` within `func()`.