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

```python

>>> X = 'iNeuron'

>>> def func():

... print(X)

>>> func()

```

The result of the code is:

```

iNeuron

```

Explanation:

- The variable `X` is defined in the global scope and has the value `'iNeuron'`.

- The function `func()` simply prints the value of `X`, which is the global `X` since the function does not have a local variable with the same name.

- When `func()` is called, it accesses the global `X` and prints its value, which is `'iNeuron'`.

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

```python

>>> X = 'iNeuron'

>>> def func():

... X = 'NI!'

>>> func()

>>> print(X)

```

The result of the code is:

```

iNeuron

```

Explanation:

- The variable `X` is defined in the global scope and has the value `'iNeuron'`.

- Inside the `func()` function, a new local variable `X` is created and assigned the value `'NI!'`. This local `X` shadows the global `X` inside the function.

- When `func()` is called, the local `X` is modified, but the global `X` remains unchanged.

- After `func()` is called, the `print(X)` statement accesses the global `X`, which still has the value `'iNeuron'`.

3. \*\*What does this code print, and why?\*\*

```python

>>> X = 'iNeuron'

>>> def func():

... X = 'NI'

... print(X)

>>> func()

>>> print(X)

```

The output of the code is:

```

NI

iNeuron

```

Explanation:

- The variable `X` is defined in the global scope and has the value `'iNeuron'`.

- Inside the `func()` function, a new local variable `X` is created and assigned the value `'NI'`. This local `X` shadows the global `X` inside the function.

- When `func()` is called, it prints the value of the local `X`, which is `'NI'`.

- After `func()` is called, the `print(X)` statement accesses the global `X`, which still has the value `'iNeuron'`.

4. \*\*What output does this code produce? Why?\*\*

```python

>>> X = 'iNeuron'

>>> def func():

... global X

... X = 'NI'

>>> func()

>>> print(X)

```

The output of the code is:

```

NI

```

Explanation:

- The variable `X` is defined in the global scope and has the value `'iNeuron'`.

- Inside the `func()` function, the `global X` statement is used to indicate that the function should operate on the global `X` variable, rather than creating a new local variable.

- When `func()` is called, it modifies the global `X` variable, changing its value to `'NI'`.

- After `func()` is called, the `print(X)` statement accesses the global `X`, which now has the value `'NI'`.

5. \*\*What about this code—what's the output, and why?\*\*

```python

>>> X = 'iNeuron'

>>> def func():

... X = 'NI'

... def nested():

... print(X)

... nested()

>>> func()

>>> X

```

The output of the code is:

```

NI

'iNeuron'

```

Explanation:

- The variable `X` is defined in the global scope and has the value `'iNeuron'`.

- Inside the `func()` function, a new local variable `X` is created and assigned the value `'NI'`. This local `X` shadows the global `X` inside the function.

- The `nested()` function is defined within `func()` and has access to the local `X` variable from the enclosing `func()` function. When `nested()` is called, it prints the value of the local `X`, which is `'NI'`.

- After `func()` is called, the final line `>>> X` simply returns the value of the global `X`, which is still `'iNeuron'`.

6. \*\*How about this code: what is its output in Python 3, and explain?\*\*

```python

>>> def func():

... X = 'NI'

... def nested():

... nonlocal X

... X = 'Spam'

... nested()

... print(X)

>>> func()

```

The output of the code in Python 3 is:

```

Spam

```

Explanation:

- The `func()` function defines a local variable `X` with the value `'NI'`.

- Inside the `nested()` function, the `nonlocal X` statement is used to indicate that the function should operate on the `X` variable from the enclosing `func()` function, rather than creating a new local variable.

- When `nested()` is called, it modifies the `X` variable from the enclosing `func()` function, changing its value to `'Spam'`.

- After `nested()` is called, the `print(X)` statement within `func()` prints the modified value of `X`, which is now `'Spam'`.

The use of the `nonlocal` keyword is a Python 3 feature that allows a nested function to access and modify variables from the enclosing scopes, without having to use global variables. This provides a way to create closures and work with variables in a more controlled and modular way.