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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

Ans: The result of the code will be:

iNeuron

Explanation:

i.) In the code, a variable `X` is assigned the string value `'iNeuron'`.

ii.) Then, a function named `func()` is defined, which does not take any arguments.

iii.) Inside the function `func()`, it simply prints the value of the variable `X`.

iv.) Finally, the function `func()` is called.

When the function `func()` is called, it prints the value of the variable `X`, which is `'iNeuron'`. Therefore, the output will be `'iNeuron'`.

The reason why the function is able to access the variable `X` even though it's defined outside the function is that in Python, functions have access to variables defined in the global scope (i.e., outside the function) unless a variable with the same name is defined within the function. This concept is known as "variable scoping" in Python.

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

Ans: The result of the code will be:

iNeuron

Explanation:

a.) In the code, a variable `X` is assigned the string value `'iNeuron'`.

b.) Then, a function named `func()` is defined, which does not take any arguments.

c.) Inside the function `func()`, a local variable `X` is assigned the string value `'NI!'`. This local variable `X` is distinct from the global variable `X` defined outside the function. It creates a new local scope within the function.

d). Next, the function `func()` is called, but it doesn't explicitly return any value.

e.) After calling the function, the `print(X)` statement outside the function is executed.

Since the variable `X` inside the function is local to that function and does not affect the global variable `X`, the global variable `X` remains unchanged and retains its value of `'iNeuron'`. Therefore, when `print(X)` is executed, it outputs the value of the global variable `X`, which is `'iNeuron'`. The local variable `X` defined inside the function has no impact on the value of the global variable `X`.

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

Ans: The code will print:

NI

iNeuron

Explanation:

a. In the code, a variable `X` is assigned the string value `'iNeuron'`.

b. Then, a function named `func()` is defined, which does not take any arguments.

c. Inside the function `func()`, a local variable `X` is assigned the string value `'NI'`. This local variable `X` is distinct from the global variable `X` defined outside the function. It creates a new local scope within the function.

d. Next, the `print(X)` statement inside the function is executed. Since `X` is a local variable within the function, it will print the value of the local variable, which is `'NI'`.

e. After calling the function, the `print(X)` statement outside the function is executed.

Since the `print(X)` statement outside the function is not inside any function and is in the global scope, it prints the value of the global variable `X`, which is `'iNeuron'`. The local variable `X` defined inside the function does not affect the value of the global variable `X`, and they are completely separate variables. So, the first `print(X)` inside the function prints `'NI'`, and the second `print(X)` outside the function prints `'iNeuron'`.

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

Ans: The output of this code will be:

NI

Explanation:

a. In the code, a variable `X` is assigned the string value `'iNeuron'`.

b. Then, a function named `func()` is defined, which does not take any arguments.

c. Inside the function `func()`, the `global` keyword is used to declare the variable `X` as a global variable. This means that any changes made to `X` within the function will affect the global variable `X` defined outside the function.

d. Next, inside the function, the value of the global variable `X` is updated to `'NI'`.

e. After calling the function, the `print(X)` statement outside the function is executed.

Since the function `func()` modifies the global variable `X` using the `global` keyword, the value of the global variable `X` is updated to `'NI'`. When `print(X)` is executed after calling the function, it prints the updated value of the global variable `X`, which is `'NI'`. The modification made to `X` inside the function affects the value of the global variable `X`, and the final output is `'NI'`.

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 output of this code will be:

NI

'iNeuron'

Explanation:

I. In the code, a variable `X` is assigned the string value `'iNeuron'`. Then, a function named `func()` is defined, which does not take any arguments. Inside the function `func()`, a local variable `X` is assigned the string value `'NI'`. This local variable `X` is distinct from the global variable `X` defined outside the function `func()`. It creates a new local scope within the function. Inside the function `func()`, another nested function named `nested()` is defined. Inside the `nested()` function, it tries to print the value of `X`. However, since `X` is not defined within the `nested()` function's scope, Python will look for it in the enclosing scope, which is the `func()` function. Python finds the local variable `X` defined in the `func()` function, and it prints the value of the local variable, which is `'NI'`.

After defining the nested function, `func()` is called. When `func()` is called, it calls the nested function `nested()`. Inside `nested()`, it prints the value of the local variable `X`, which is `'NI'`.The first print statement inside the nested function will output `'NI'`.After the function calls are completed, the code encounters the last line of the code: `X`.When `X` is referenced without being in any function scope, it refers to the global variable `X` defined outside the `func()` function. Therefore, the second print statement will output the value of the global variable `X`, which is `'iNeuron'`.

Hence, the output will be:

NI

'iNeuron'

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: In Python 3, the output of the provided code will be:

Spam

Explanation:

In the code, there are two functions: `func()` and `nested()`. Inside the `func()` function, a local variable `X` is assigned the string value `'NI'`.Inside the `nested()` function, it uses the `nonlocal` keyword to indicate that the variable `X` being referenced is not local to the `nested()` function, but it is from the nearest enclosing scope, which is the `func()` function.The `nested()` function then reassigns the value of the `nonlocal` variable `X` to `'Spam'`.After defining the functions, the `func()` function is called.When `func()` is called, it executes the nested function `nested()`.Inside `nested()`, it modifies the value of the `nonlocal` variable `X` to `'Spam'`.

Since the variable `X` is marked as `nonlocal`, the change made inside the `nested()` function affects the variable `X` in the enclosing `func()` function. The `nested()` function completes its execution, and the control returns to the `func()` function. After the nested function call, the `print(X)` statement in the `func()` function is executed.The `print(X)` statement will print the value of the modified `nonlocal` variable `X`, which is `'Spam'`.

Hence, the output will be:

Spam