**ADITYA AMIN**  **ASSIGN : 22**

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

>>> X = 'iNeuron'

>>> def func():

print(X)

>>> func()

Output :

'iNeuron'

Explanation:

The global variable X is defined with the value 'iNeuron'.

The function func() is called, which does not take any arguments.

Inside the function body, the print() statement prints the value of the global variable X, which is 'iNeuron'.

Therefore, when the function is called, it prints 'iNeuron'.

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

>>> X = 'iNeuron'

>>> def func():

X = 'NI!'

>>> func()

>>> print(X)

Output :

'iNeuron'

Explanation:

The global variable X is defined with the value 'iNeuron'.

The function func() is called, which does not take any arguments.

Inside the function body, a local variable X is assigned the value 'NI!'. However, this does not affect the global variable X because local variables are only accessible within the function scope.

The function finishes executing, and then the print() statement prints the value of the global variable X, which is 'iNeuron'.

Therefore, after calling the function and printing the value of X, it prints 'iNeuron', as the global variable X was not modified by the function.

3. What does this code print, and why?

>>> X = 'iNeuron'

>>> def func():

X = 'NI'

print(X)

>>> func()

>>> print(X)

Output will be :

'iNeuron'

Explanation:

The global variable X is defined with the value 'iNeuron'.

The function func() is called, which does not take any arguments.

Inside the function body, a local variable X is assigned the value 'NI'. This local variable is only accessible within the function scope and does not affect the global variable X.

The print() statement inside the function body prints the value of the local variable X, which is 'NI'.

The function finishes executing, and then the print() statement outside the function body prints the value of the global variable X, which is 'iNeuron'.

Therefore, after calling the function and printing the values of X before and after the function call, it prints 'NI' and 'iNeuron', respectively.

4. What output does this code produce? Why?

>>> X = 'iNeuron'

>>> def func():

global X

X = 'NI'

>>> func()

>>> print(X)

Output will be :

'NI'

Explanation:

The global variable X is defined with the value 'iNeuron'.

The function func() is called, which does not take any arguments.

Inside the function body, the global keyword is used to indicate that the variable X being accessed and modified is a global variable, not a local variable. This means that the assignment X = 'NI' inside the function modifies the global variable X, changing its value to 'NI'.

The function finishes executing, and then the print() statement outside the function body prints the value of the global variable X, which is now 'NI'.

Therefore, after calling the function and printing the value of X, it prints '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

Output will be :

'NI'

'iNeuron'

Explanation:

The global variable X is defined with the value 'iNeuron'.

The function func() is called, which defines a local variable X inside its function body and assigns it the value 'NI'.

Inside the function func(), another function nested() is defined, which attempts to print the value of the local variable X. Since nested() is defined within func(), it has access to the local variables of its enclosing scope, which includes the local variable X of func(). Therefore, it prints 'NI'.

The function nested() is called from func(), and it prints 'NI'.

After the function nested() finishes executing, the function func() finishes executing as well.

Finally, the X variable is printed outside the function func(), and it prints the global variable X, which is still 'iNeuron'.

Therefore, the code prints 'NI' and '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()

Output :

SyntaxError: no binding for nonlocal 'X' found

Explanation:

The nonlocal keyword is used inside the function nested() to indicate that the variable X being referred to is not local to nested(), but rather refers to the nearest enclosing scope outside of nested().

However, in this case, there is no such enclosing scope that contains a variable X. This is because nested() is defined inside func(), which itself defines a local variable X.

As a result, when the code is executed in Python 3, it raises a SyntaxError with the message "no binding for nonlocal 'X' found". This error occurs because nonlocal can only be used to refer to variables in an enclosing scope, and in this case, there is no enclosing scope that matches the variable X referred to in nested().

To fix this error, you can either remove the nonlocal keyword and treat X as a local variable in nested(), or you can define an appropriate enclosing scope that contains a variable X that nested() can refer to using the nonlocal keyword.