ASSIGNMENT - 22 (PYTHON)

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

```
>>> X = 'iNeuron'
>>> def func():
print(X)

>>> func()
```

The provided code defines a variable X with the value 'iNeuron' and a function func() that prints the value of X.

Let's break down the code execution:

X = 'iNeuron': This line assigns the string 'iNeuron' to the variable X.
def func():: This line defines a function named func.
print(X): Inside the func() function, it prints the value of the variable X.
When func() is called:

```
func()
```

It prints the value of X, which is 'iNeuron'.
So, the result of the code would be: 'iNeuron'

Explanation: The function func() prints the value of the 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)
```

The provided code defines a variable X with the value 'iNeuron' in the global scope, and then defines a function func(). Inside func(), a local variable X is defined with the value 'NI!', but it is not used. After calling func(), the value of the global variable X is printed.

Let's break down the code execution:

X = 'iNeuron': This line assigns the string 'iNeuron' to the variable X in the global scope.
def func():: This line defines a function named func.
X = 'NI!': Inside the func() function, it defines a local variable X with the value 'NI!'. However, this local variable is not used or returned by the function.
func(): This line calls the function func(). Inside the function, a local variable X is defined, but it doesn't affect the value of the global variable X.
print(X): After calling func(), it prints the value of the global variable X.
When print(X) is executed after func():print(X)

It prints the value of the global variable X, which remains unchanged as 'iNeuron'.

So, the result of the code would be: iNeuron

Explanation: Even though a local variable X is defined inside the function func(), it doesn't affect the value of the global variable X. Therefore, when print(X) is executed, it prints the value of the global variable X, which is 'iNeuron'.

3. What does this code print, and why?

```
>>> X = 'iNeuron'
>>> def func():
X = 'NI!'
print(X)


>>> func()
>>>print(X)
```

The provided code defines a variable X with the value 'iNeuron' in the global scope and then defines a function func(). Inside func(), a local variable X is defined with the value 'NI!', and it is printed within the function. After calling func(), the value of the global variable X is printed.

Let's break down the code execution:

X = 'iNeuron': This line assigns the string 'iNeuron' to the variable X in the global scope.
def func():: This line defines a function named func.
X = 'NI!': Inside the func() function, it defines a local variable X with the value 'NI!'.
print(X): Inside the func() function, it prints the value of the local variable X, which is 'NI!'.
func(): This line calls the function func(). Inside the function, a local variable X is defined and printed, but it doesn't affect the value of the global variable X.
print(X): After calling func(), it prints the value of the global variable X.
When print(X) is executed after func(): print(X)

It prints the value of the global variable X, which remains unchanged as 'iNeuron'.

So, the result of the code would be:
NI!
iNeuron

Explanation: Inside the func() function, a local variable X is defined with the value 'NI!', and it is printed within the function. However, this local variable X is different from the global variable X. After calling func(), when print(X) is executed, it prints the value of the global variable X, which is 'iNeuron'.

4. What output does this code produce? Why?
>>> X = 'iNeuron'
>>> def func():
global X
X = 'NI!'

>>> func()
>>>print(X)

This code defines a global variable X with the value 'iNeuron' and then defines a function func() which modifies the global variable X using the global keyword. After calling func(), it prints the value of the global variable X.

Let's break down the code execution:

X = 'iNeuron': This line assigns the string 'iNeuron' to the variable X in the global scope.
def func():: This line defines a function named func.
global X: Inside the func() function, it declares that the variable X being referenced is the global variable X.
X = 'NI!': Inside the func() function, it assigns the value 'NI!' to the global variable X.
func(): This line calls the function func(), which modifies the global variable X to 'NI!'.
print(X): After calling func(), it prints the value of the global variable X.
When print(X) is executed after func(): print(X)

It prints the modified value of the global variable X, which is 'NI!'.

So, the output of the code would be: NI!

Explanation: Inside the func() function, the global keyword is used to declare that the variable X being referenced is the global variable X. Therefore, when X is assigned the value 'NI!' inside

the function, it modifies the global variable X. After calling func(), print(X) prints the modified value of the global variable 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
```

In this code, a global variable X is defined with the value 'iNeuron', and a function func() is defined. Inside func(), another function nested() is defined. Inside nested(), it prints the value of X. After defining the functions, func() is called, which in turn calls nested(). Finally, the value of X is printed.

Let's break down the code execution:

X = 'iNeuron': This line assigns the string 'iNeuron' to the variable X in the global scope.
def func():: This line defines a function named func.
X = 'NI!': Inside the func() function, it defines a local variable X with the value 'NI!'. However, this local variable X is not used or returned by the function.
def nested():: Inside the func() function, it defines another function named nested.
print(X): Inside the nested() function, it prints the value of the variable X, which is the local variable X defined in the func() function. Since nested() is called within func(), it has access to the local variables of func().
nested(): After defining the functions, func() is called, which in turn calls nested(). Inside nested(), it prints the value of X.
After calling func(), the value of X is printed.
When X is printed after func(): print(X)

It prints the value of the global variable X, which remains unchanged as 'iNeuron'.

So, the output of the code would be: iNeuron

Explanation: Inside the nested() function, it prints the value of the local variable X, which is defined in the func() function. After calling func(), the value of the global variable X is printed, which is '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()
```

In Python 3, the provided code would raise a syntax error because the nonlocal keyword is used to declare that a variable is not local to the current function, but it is also not a global variable. However, in the given code, X is not defined as nonlocal in the func() function, which causes the syntax error.

Here's the corrected version of the code:

```
def func():
    X = 'NI!'
    def nested():
        nonlocal X
        X = 'Spam'
    nested()
    print(X)

func()
```

With the corrected version, the output of the code would be:
Spam

Explanation: In the corrected code, the nonlocal keyword is used inside the nested() function to declare that the variable X is not local to nested() but is also not a global variable. Therefore, when nested() is called within func(), it modifies the value of X to 'Spam'. After calling nested(), print(X) prints the modified value of X, which is 'Spam'.