Name :- Deepawali . B. Mhaisagar
Assignment no 22

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

```
>>> X = ‘iNeuron’
>>>def func():
print(X)

>>> func()
```

ANS
The code will print the value of the variable `X`, which is `'iNeuron'`.

Here's an explanation of the code:

1. First, the variable `X` is assigned the string value `'iNeuron'`.
2. Next, a function named `func` is defined.
3. Inside the `func` function, the value of the variable `X` is printed using the `print()` function.
4. Finally, the `func()` function is called.

When the `func()` function is called, it prints the value of the variable `X`, which is `'iNeuron'`. This happens because the function can access and reference variables from the global scope (outside the function). In this case, `X` is defined outside the `func()` function, so it is accessible and can be printed within the function.

Therefore, the output of the code will be:

```
iNeuron
```

It will print the string `'iNeuron'` as the result.

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'` when `print(X)` is executed.

Here's an explanation of the code:

1. First, the variable `X` is assigned the string value `'iNeuron'`.
2. Then, a function named `func` is defined.
3. Inside the `func` function, a new local variable `X` is assigned the string value `'NI!'`. This local variable `X` shadows the global variable `X` within the scope of the `func` function. It does not affect the value of the global `X` variable.
4. The `func` function is called, but since there is no print statement or return statement within the function, it does not produce any visible output.
5. Finally, the `print(X)` statement is executed outside the `func` function.

When `print(X)` is executed outside the `func` function, it refers to the global variable `X` and not the local variable `X` within the function. As a result, it prints the value of the global variable `X`, which is `'iNeuron'`.

Therefore, the output of the code will be:

```
iNeuron
```

It will print the string `'iNeuron'` as the result, confirming that the global variable `X` remains unaffected by the local variable assignment within the `func` function.

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'` and then `'iNeuron'`.

Here's an explanation of the code:

1. First, the variable `X` is assigned the string value `'iNeuron'`.
2. Then, a function named `func` is defined.
3. Inside the `func` function, a new local variable `X` is assigned the string value `'NI'`. This local variable `X` shadows the global variable `X` within the scope of the `func` function.
4. The `print(X)` statement within the `func` function prints the value of the local variable `X`, which is `'NI'`.
5. The `func()` function is called.
6. After the function call, the `print(X)` statement is executed outside the `func` function.
7. When `print(X)` is executed outside the `func` function, it refers to the global variable `X` and not the local variable `X` within the function. As a result, it prints the value of the global variable `X`, which is `'iNeuron'`.

Therefore, the output of the code will be:

```
NI
iNeuron
```

It first prints `'NI'` as the value of the local variable `X` within the `func` function. Then, it prints `'iNeuron'` as the value of the global variable `X` outside the `func` function.

4. What output does this code produce? Why?

>>> X = ‘iNeuron&’
>>> def func():
global X
X = ‘NI’

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

The output of the code will be `'NI'`.

Here's an explanation of the code:

1. First, the variable `X` is assigned the string value `'iNeuron&'`.
2. Then, a function named `func` is defined.
3. Inside the `func` function, the `global` keyword is used to declare that the variable `X` is a global variable, not a local variable. This means that any modifications to `X` inside the function will affect the global `X` variable.
4. The line `X = 'NI'` inside the `func` function assigns the string value `'NI'` to the global variable `X`, modifying its value.
5. The `func()` function is called.
6. After the function call, the `print(X)` statement is executed outside the `func` function.
7. When `print(X)` is executed outside the `func` function, it refers to the modified global variable `X` which was updated within the function. Therefore, it prints the value `'NI'`.

Therefore, the output of the code will be:

```
NI
```

It prints `'NI'` as the result since the global variable `X` was modified within the `func` function using the `global` keyword.

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 code will output `'NI'` and then `'iNeuron'`.

Here's an explanation of the code:

1. First, the variable `X` is assigned the string value `'iNeuron'`.
3. The `func` function is defined.
4. Inside the `func` function, a new local variable `X` is assigned the string value `'NI'`. This local variable `X` shadows the global variable `X` within the `func` function.
5. Then, a nested function named `nested` is defined within the `func` function.
6. Inside the `nested` function, the `print(X)` statement is executed. It refers to the local variable `X` defined in the `func` function, which is `'NI'`. Therefore, it prints `'NI'`.
7. The `nested()` function is called within the `func` function, so it is executed and prints `'NI'`.
8. After defining the `func` function, it is called using `func()`.
9. Finally, the value of the global variable `X` is printed using `X`. It refers to the global variable `X` defined outside the `func` function, which is `'iNeuron'`. Therefore, it prints `'iNeuron'`.

Therefore, the output of the code will be:

```
NI
iNeuron
```

It first prints `'NI'` as the result of the nested function call within `func()`, and then it prints `'iNeuron'` as the value of the global variable `X` outside the `func` function.

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 code will output `'Spam'`.

Here's an explanation of the code:

1. The `func` function is defined.
2. Inside the `func` function, a new local variable `X` is assigned the string value `'NI'`.
3. Then, a nested function named `nested` is defined within the `func` function.
4. Inside the `nested` function, the `nonlocal` keyword is used to declare that the variable `X` is a nonlocal variable. It means that any modifications to `X` inside the `nested` function will affect the variable `X` in the nearest enclosing scope, which is the `func` function in this case.
5. The line `X = 'Spam'` inside the `nested` function assigns the string value `'Spam'` to the nonlocal variable `X`, modifying its value.
6. The `nested()` function is called within the `func` function.
7. After the `nested()` function call, the `print(X)` statement is executed inside the `func` function.
8. When `print(X)` is executed, it refers to the nonlocal variable `X`, which was modified within the `nested` function. Therefore, it prints the value `'Spam'`.

Therefore, the output of the code will be:

```
Spam
```

It prints `'Spam'` as the result since the `nonlocal` keyword allows the modification of the nonlocal variable `X` within the nested function `nested()`, and that change is visible in the `print(X)` statement inside the `func()` function.