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

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


>>> func()
```

Answer: Output is **iNeuron**

X is a global variable containing the string ‘iNeuron, which is printed when the function func() is called

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

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


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

Answer: Output is **iNeuron**.

X is assigned a string “iNeuron’ which is a global variable. When the function func() is called X is assigned a local value ‘NI’,

When print(X) is called it prints 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)
```

Answer: The output is :-

NI
iNeuron

func() returns the local value of X which is NI and print(X) returns the global value of the variable X which is iNeuron

4. What output does this code produce? Why?

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

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

Answer: **The Output is 'NI'**

X is assigned global value 'iNeuron', however the variable is assigned as global inside func() function, which is then assigned 'NI'. Thus print(X) returns the overwritten global value '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
```

Answer: The output is:-

iNeuron

'iNeuron'

X is assigned the global value 'iNeuron'.

nested() function prints the global variable of X, which is iNeuron and X prints again the global value of X which is iNeuron. So Ineuron is printed twice

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()
```

Answer: This output is **'Spam'**,

Because X is defined as nonlocal in the nested function, its neither local nor global variable. Hence print(X) gives the value of X as the nonlocal value of X, because we are printing X after calling the inner function. However, if we write print(X) before calling nested() inner function, it will give output as 'NI'