

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

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

>>> func()
```



```
1
2 X = 'iNeuron'
3 def func():
4 print(X)
5
6
7 func()
8
```

File "<ipython-input-4-ec3bc4a84f67>", line 4  
print(X)  
^  
IndentationError: expected an indented block

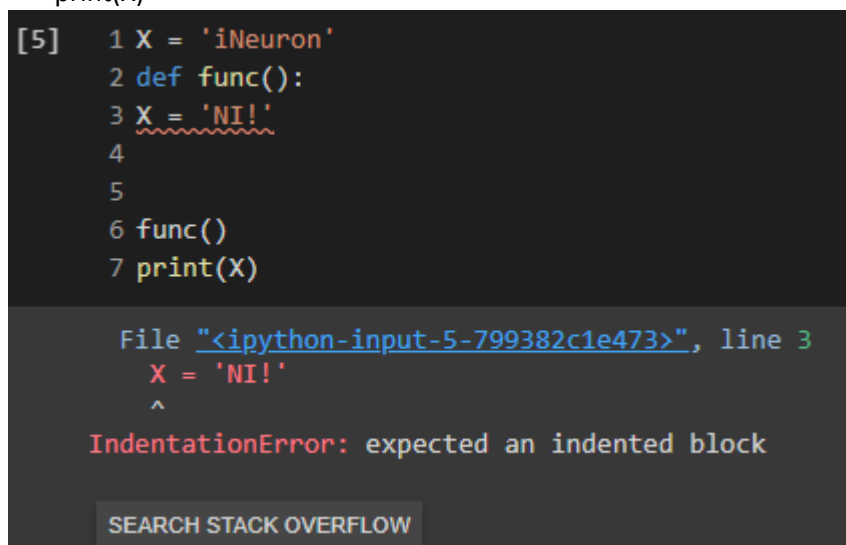
SEARCH STACK OVERFLOW

There is an indentation missing. Indentation behind the print will solve the issue.

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

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

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



```
[5] 1 X = 'iNeuron'
2 def func():
3 X = 'NI!'
4
5
6 func()
7 print(X)
```

File "<ipython-input-5-799382c1e473>", line 3  
X = 'NI!'  
^  
IndentationError: expected an indented block

SEARCH STACK OVERFLOW

There is an indentation missing & print function is outside the def function. Indentation behind the variable X in line 3 and print(X) in line 4 with an indentation will solve the issue.

3. What does this code print, and why?

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

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

It will throw an error. Because of the missing indentation in line no. 3 & 4.

After giving an indentation in each lines it'll print as below :

NI!

iNeuron

First print will be a modified variable.

Second print will print the first defined variable because it is outside of the **def** function.

```
1 X = 'iNeuron'
2 def func():
3     X = 'NI!'
4     print(X)
5
6 func()
7 print(X)

NI!
iNeuron
```

4. What output does this code produce? Why?

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

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

It will throw an error. Because of the missing indentation in line no. 3 & 4.

After giving an indentation in each lines it'll print as below :

NI!

In this program by using the **global** keyword, we can modify the local variable into a global variable. (Outside of the **def** function also the value of variable modifies)

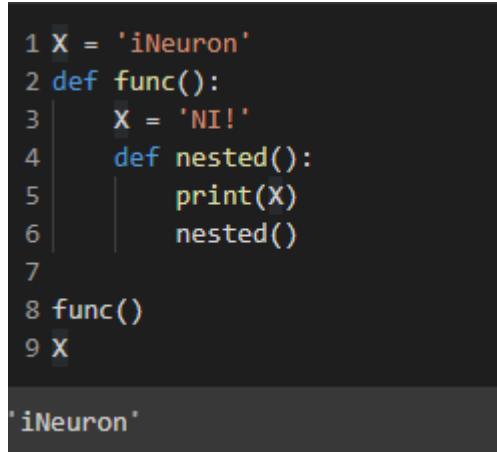
```
1 X = 'iNeuron'
2 def func():
3     global X
4     X = 'NI!'
5
6 func()
7 print(X)

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
```



```
1 X = 'iNeuron'
2 def func():
3     X = 'NI!'
4     def nested():
5         print(X)
6         nested()
7
8 func()
9 X
```

'iNeuron'

It will throw an error. Because of the missing indentation in line no. 3, 4, 5 & 6.

After giving required indentation in each lines it'll print as below :

**iNeuron**

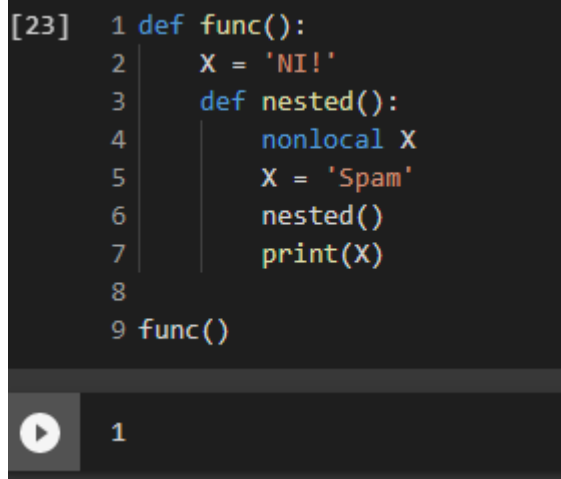
Because of the nested function X considering the global variable. Hence the output is the global variable.

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()
```

After giving the required indentation the output for the program will be nothing, because global variable is not defined in the first place & secondly by using nonlocal keyword we defined the X variable as a global variable inside the nested function.



```
[23] 1 def func():
2     X = 'NI!'
3     def nested():
4         nonlocal X
5         X = 'Spam'
6         nested()
7         print(X)
8
9 func()
```

1