Global keywords in Python

What does the global key do and how do I use it in Python? Please follow us.

In the last Python variable, when I mentioned the problem of changing the value of the global variable in a function, I said I must use the Python keyword as `global` , but I didn't say how to use it because I wanted to set aside a article to write. Details about this keyword.

What does the `global` key do and how do I use it in Python? Please follow us.

In Python, the `global` keyword allows you to edit variables outside the current scope. It is used to create global variables and make changes to variables in the local context.

Rules of global keywords in Python

1. When we create variables in a function, it defaults to local variables.
2. When we define a variable outside the function, it defaults to global variables. You do not need to use global keywords.
3. We use the `global` keyword to read and write global variables in a function.
4. Using the `global` keyword outside a function does not work at all.

Using global keywords in Python

Example 1: Access global variables from within a function

```
a = 1 # Bi?n toàn c?c

def them():
 print(a)

them()
```

When we run the above code, we get the output of 1. However, there are some cases where we need to modify the global variable from within the function, which is the case I mentioned from the beginning, so what to do?

Example 2: Modifying global variables in a function

Suppose we need to edit the value of a to a + 9 in the `them()` function, if we write the following code:

```
a = 1 # Bi?n toàn c?c
```

```
def them():
 a = a + 9
print(a)

them()
```

You will receive an error message:

```
UnboundLocalError: bi?n bi?n 'm?t' ? ??a ch? tr??c khi giao d?ch
```

That's because we can only access global variables and cannot edit them in a function. The solution to this problem is to use the `global` keyword. Then, the above code will be rewritten as follows:

```
a = 1 # Bi?n toàn c?c

def them():
 global a
a = a + 9
 print("Trong them():", a)

them()
print("Trong main:", a)
```

Running the above code we get the output as:

```
In addition (): 10
In main: 10
```

Here, we define a as a global variable in the `them()` function, then increase the value of `a` to 9, ie `a = a + 9` . Then, we call the `them()` function `them()` , finally, print global variables `a` . As a result, changes made to the variable a in the `them()` function `them()` also occur on the global variable outside the function, a = 10.

Example 3: Share global global variables via modules in Python

In Python, we create a dun config.py to hold global variables and share information through Python modules in the same program. This is how we can share global variables through Python modules.

Create a config.py file to store global variables:

```
a = 0
b = "empty"
```

Create an update.py file to change global variables:

```
import config

config.a = 10
config.b = "TipsMake.com"
```

Write the main.py file to check for changes:

```
import config
import update
```

```
print (config.a)
print (config.b)
```

When running main.py file, the output will be:

```
ten
TipsMake.com
```

Here, we have created 3 files, config.py, update.py and main.py. The config.py module stores two global variables, `a` and `b` . In the update.py file we enter the module config.py and modify the value of the variable `a, b` . Similar to the main.py file, enter both modules config.py and update.py. Finally, we use the print command to check if the values ??of variables a and b have been changed.

Example 4: Use global variables in nested functions.

In this example you will know how to use global variables in nested functions.

```
def ham1 ():
x = 20

def ham2 ():
global x
x = 25

print ("Before calling ham2:", x)
print ("Calling ham2")
ham2 ()
print ("After calling ham2:", x)

ham1 ()
print ("x in main:", x)
```

Running the above code we can:

```
Before calling ham2: 20
Calling ham2
After calling ham2: 20
x in main: 25
```

Here, we declare the global variable in the nest function `ham2()` . In `ham1()` , `x` not affected by `global` keywords.

Before and after calling `ham2()` , `x` will take the value of the local variable `20` . Outside the `ham1(), x` function `ham1(), x` will take the global value, reported in `ham2() 25` . This is because we use the `global` keyword in `x` to create global variables in `ham2()` . If we make any changes to `x` in `ham2()` , the change will appear outside the local scope.

Do a Python exercise with a prize to practice more.

Next lesson: Module in Python

Previous post: Global variable (global), local variable (local), nonlocal variable in Python