How to use Closure in Python

In this article, TipsMake.com will work with you to learn about Closure in Python, how to define a Closure and why you should use it. Let's go find the answer!

In this article, TipsMake.com will work with you to learn about Closure in Python, how to define a Closure and why you should use it. Let's go find the answer!

Nonlocal variable in nested function

Before going to learn what Closure is, go through the nested function and turn on the nonlocal a bit.

A function defined inside another function is called a nested function. Nested functions can access variables and return the results of functions in the enclosed scope.

In Python, the nonlocal variable is used in nested functions where local scope is not defined. Understandably, the nonlocal variable is not a local variable, not a global variable, you declare a variable as nonlocal when you want to use it at a wider range than local, but not to a global level.

1. See also: Global variable (global), local variable (local), nonlocal variable in Python

Example: The nested access function turns nonlocal

```
 def print_msg(msg): # Hàm bên ngoài def printer(): # Hàm l?
ng nhau print(msg) printer() # Ch?y hàm # Output: Hello print_msg("Hello")
```

We can see that the nested *printer* function *()* can access the nonlocal *msg* variable of the function outside it.

Defining Closure function in Python

Closure is usually created by nested functions, which can be interpreted as functions to remember the space where it creates.

```
 def print_msg(msg): # Hàm bê ngoài def printer(): # Hàm l?
ng nhau print(msg) return printer another = print_msg("Hello") another()
```

In the above example, when *print_msg* is called with the "Hello" string, the child printer function will be defined with the value of the local *msg* . After that, *another ()* function is called. When we execute the function, because the *printer* references *msg* , this value will be remembered even if the *print_msg* function has ended before.

Run them in Python shell to see output:

```
>>> del print_msg >>> another() Hello >>> print_msg("Hello") Traceback (most rec
```

It should be noted that closures are created by nested functions but not all nested functions are closure. Closure is only created when the child function accesses local variables in the scope closed by the parent function.

Conditions for having Closure

As in the above example, we see Closure used in Python when a nested function references a value within scope, where it is defined.

Criteria for creating Closure in Python must be met, summarized in the following points.

To create closure in Python, the criteria that need to be met include:

1. There is a nested function (function defined inside another function).
2. The nested function must refer to a value defined in the enclosed function.
3. The enclosed function must return the result to the nested function.

When should I use Closure?

Closure can use global values ??and provide some form of hidden data. Closure can also provide solutions, creating functions that have many properties of object-oriented programming to solve the problem.

When a class needs to be defined with only a few methods, a closure can be used as a lighter alternative to object-oriented programming. However, when properties and methods increase, using object-oriented programming will be a better solution.

Here is a simple example in which closing may be more appropriate than defining a class and creating objects. But the hobby is all yours.

The example below will show you the use of closures more appropriately than the object-oriented programming method with class definition and object creation. However, depending on how you want, use any.

```
def make_multiplier_of(n): def multiplier(x): return x * n return multiplier # H
```

In applications of nested functions, closure is the most important application. If we know how to use it properly, we will write beautiful code, high performance and easy maintenance.

Stay tuned to TipsMake.com's Python lesson series to learn more about this programming language!

Previous article: Generator in Python

Next lesson: Decorator in Python

guides. Thank you for reading and for following us regularly.