Python TUTORIAL

Python nonlocal

Summary: in this tutorial, you'll learn about the Python nonlocal scopes and how to use the `nonlocal` keyword to change the variables of the nonlocal scopes.

Introduction to Python nonlocal scopes

In Python, you can define a function (https://www.pythontutorial.net/python-basics/python-functions/) inside another function. For example:

```
def outer():
    print('outer function')

    def inner():
        print('inner function')

    inner()
```

```
outer()
```

Output:

```
outer function
inner function
```

In this example, we defined a function called `outer` .

Inside the `outer` function, we defined another function called `inner` . And we called the `inner` function from the inside of the `outer` function.

Often, we say that the `inner` function is nested in the `outer` function. In practice, you define nested functions when you don’t want these functions to be global.

Both `outer` and `inner` have access to the global and built-in scopes (https://www.pythontutorial.net/advanced-python/python-variable-scopes/) as well as their local scopes.

And the `inner` function also has access to its enclosing scope, which is the scope of the `outer` function.

From the `inner()` function perspective, its enclosing scope is neither local nor global. And Python calls this a nonlocal scope.

Let’s modify the `outer` and `inner` functions:

```
def outer():
    message = 'outer function'
    print(message)

    def inner():
        print(message)

    inner()
```

```
outer()
```

Output:

```
outer function
outer function
```

When we call the `outer` function, Python creates the `inner` function and executes it.

When the `inner` function executes, Python doesn't find the `message` variable in the local scope. So Python looks for it in the enclosing scope, which is the scope of the `outer` function:

See the following example:

```
message = 'global scope'
```

```
def outer():

    def inner():
        print(message)

    inner()


outer()
```

Output:

```
global scope
```

In this example, Python searches for the `message` variable in the local scope of the `inner` function.

Since Python doesn’t find the variable, it searches for the variable in its enclosing scope, which is the scope of the `outer` function.

And in this case, Python goes up to the global scope to find the variable:

Python nonlocal keyword

To modify variables from a nonlocal scope in a local scope, you use the `nonlocal` keyword. For example:

```
def outer():
    message = 'outer scope'
    print(message)

    def inner():
        nonlocal message
        message = 'inner scope'
        print(message)

    inner()

    print(message)
```

```
outer()
```

Output:

```
outer scope
inner scope
inner scope
```

In this example, we use `nonlocal` keyword to explicitly instruct Python that we're modifying a nonlocal variable.

When you use the `nonlocal` keyword for a variable, Python will look for the variable in the enclosing local scopes chain until it first encounters the variable name.

More importantly, Python won't look for the variable in the global scope.

Consider the following example:

```
message = 'outer scope'


def outer():
    print(message)

    def inner():
        nonlocal message
        message = 'inner scope'
        print(message)

    inner()

    print(message)
```

```
outer()
```

If you run this code, you'll get the following error:

```
SyntaxError: no binding for nonlocal 'message' found
```

From inside of the `inner` function, we use the `nonlocal` keyword for the `message` variable.

Therefore, Python searches for the `message` variable in the enclosing scope, which is the scope of the `outer` function.

Since the scope of the `outer` function doesn't have `message` variable and Python doesn't look further in the global scope, it issues an error:

Summary

- The enclosing scopes of inner functions are called nonlocal scopes.
- Use the `nonlocal` keyword to modify the variable from the nonlocal scopes.
- And Python will look up the nonlocal variables in the enclosing local scopes chain. It won’t search for the variable in the global scope.