名字空间

C 中，变量名是内存地址的别名，而在 Python 中，名字是一个字符串对象，它与他指向的对象构成一个{name:object}关联。
Python 由很多名字空间，而 LEGB 则是名字空间的一种查找规则。

作用域

Python 中`name-object`的关联存储在不同的作用域中，各个不同的作用域是相互独立的

简述

简而言之，LEGB 代表名字查找顺序: `locals -> enclosing function -> globals -> __builtins__`

- locals 是函数内的名字空间，包括局部变量和形参
- enclosing 外部嵌套函数的名字空间（闭包中常见）
- globals 全局变量，函数定义所在模块的名字空间
- builtins 内置模块的名字空间

当然，因为 builtins 的特殊性，我们可以直接在 builtins 里面添加变量，这样就可以在任意模块中访问变量，不过这种方法太过于变态，不推荐这么做。

locals，globals

函数的形参跟内部变量都存储在 locals 中。

```
In [1]: def f(x):
   ...:     a = x
   ...:     print a
   ...:     print locals()
   ...:


In [2]: f("hello")
hello
{'a': 'hello', 'x': 'hello'}
```

不过在函数内部调用global 声明的时候，可以将变量存储在 globals 中

```
In [6]: def f(x):
   ...:     global a
   ...:     a = x
   ...:     print a
```

```
   ...:     print locals()
   ...:

In [7]: f("hello")
hello
{'x': 'hello'}

In [8]: print a
hello

In [9]: print x
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
<ipython-input-9-2d264e11d975> in <module>()
----> 1 print x

NameError: name 'x' is not defined
```

如上面栗子中那样，在函数中声明 a 为全局变量，则函数 f 的 locals只有参数 x，而没有变量，而在外部可以使用变量 a，而使用 x 的时候则是`NameError`

Enclosed

Enclosing 是外部嵌套函数的名字空间。我们经常在闭包中用到。在 Python3中提供了一个nonlocal关键字来修改外部嵌套函数的名字空间，但是要使用 Python3才有，我等使用Python2的只能眼馋一下。

```
In [11]: def outer():
   ....:     a_var = 'enclosed value'
   ....:     print a_var
   ....:     def inner():
   ....:         a_var = 'local value'
   ....:         print(a_var)
   ....:     inner()
   ....:     print a_var
   ....:

In [12]: outer()
enclosed value
local value
enclosed value
```

下面的栗子简单示范一下 nonlocal 的用法，实在 Python3下面才可以正常运行的：

```
In [1]: a_var = 'global value'

In [2]: def outer():
   ...:     a_var = "local value"
```

```
   ...:     print("outer befor", a_var)
   ...:     def inner():
   ...:         nonlocal a_var
   ...:         a_var = "inner value"
   ...:         print("in inner():", a_var)
   ...:     inner()
   ...:     print("outer inner:", a_var)
   ...:

In [3]: outer()
outer befor local value
in inner(): inner value
outer inner: inner value

In [4]: print(a_var)
global value
```

builtins

builtins 则是内置模块，轻易不要修改

```
In [19]: b
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-19-3b5d5c371295> in <module>()
----> 1 b

NameError: name 'b' is not defined

In [20]: __builtins__.b = "builtins"

In [21]: b
Out[21]: 'builtins'
```

上面栗子中在第一次调用b的时候报错NameError，之后我们修改 builtins 的名字空间，将名字b与值"builtins"进行关联，就可以正常调用了。这种非常规用法不建议使用。