1.11 命名切片¶

问题¶

如果你的程序包含了大量无法直视的硬编码切片，并且你想清理一下代码。

解决方案¶

假定你要从一个记录（比如文件或其他类似格式）中的某些固定位置提取字段：

```
######    0123456789012345678901234567890123456789012345678901234567890'
record = '.....................100 .......513.25 ..........'
cost = int(record[20:23]) * float(record[31:37])
```

与其那样写，为什么不想这样命名切片呢：

```
SHARES = slice(20, 23)
PRICE = slice(31, 37)
cost = int(record[SHARES]) * float(record[PRICE])
```

在这个版本中，你避免了使用大量难以理解的硬编码下标。这使得你的代码更加清晰可读。

讨论¶

一般来讲，代码中如果出现大量的硬编码下标会使得代码的可读性和可维护性大大降低。 比如，如果你回过来看看一年前你写的代码，你会摸着脑袋想那时候自己到底想干嘛啊。 这是一个很简单的解决方案，它让你更加清晰的表达代码的目的。

内置的 `slice()` 函数创建了一个切片对象。所有使用切片的地方都可以使用切片对象。比如：

```
>>> items = [0, 1, 2, 3, 4, 5, 6]
>>> a = slice(2, 4)
>>> items[2:4]
[2, 3]
>>> items[a]
[2, 3]
>>> items[a] = [10,11]
>>> items
[0, 1, 10, 11, 4, 5, 6]
>>> del items[a]
>>> items
[0, 1, 4, 5, 6]
```

如果你有一个切片对象a，你可以分别调用它的 `a.start` , `a.stop` , `a.step` 属性来获取更多的信息。比如：

```
>>> a = slice(5, 50, 2)
>>> a.start
5
>>> a.stop
50
>>> a.step
2
```

```
>>>
```

另外，你还可以通过调用切片的 `indices(size)` 方法将它映射到一个已知大小的序列上。 这个方法返回一个三元组 `(start, stop, step)` ，所有的值都会被缩小，直到适合这个已知序列的边界为止。 这样，使用的时就不会出现 `IndexError` 异常。比如：

```
>>> s = 'HelloWorld'
>>> a.indices(len(s))
(5, 10, 2)
>>> for i in range(*a.indices(len(s))):
...     print(s[i])
...
W
r
d
>>>
```