8.1 改变对象的字符串显示¶

问题¶

你想改变对象实例的打印或显示输出，让它们更具可读性。

解决方案¶

要改变一个实例的字符串表示，可重新定义它的 `__str__()` 和 `__repr__()` 方法。例如：

```
class Pair:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return 'Pair({0.x!r}, {0.y!r})'.format(self)

    def __str__(self):
        return '({0.x!s}, {0.y!s})'.format(self)
```

`__repr__()` 方法返回一个实例的代码表示形式，通常用来重新构造这个实例。 内置的 `repr()` 函数返回这个字符串，跟我们使用交互式解释器显示的值是一样的。 `__str__()` 方法将实例转换为一个字符串，使用 `str()` 或 `print()` 函数会输出这个字符串。比如：

```
>>> p = Pair(3, 4)
>>> p
Pair(3, 4) # __repr__() output
>>> print(p)
(3, 4) # __str__() output
>>>
```

我们在这里还演示了在格式化的时候怎样使用不同的字符串表现形式。 特别来讲， `!r` 格式化代码指明输出使用 `__repr__()` 来代替默认的 `__str__()` 。 你可以用前面的类来试着测试下：

```
>>> p = Pair(3, 4)
>>> print('p is {0!r}'.format(p))
p is Pair(3, 4)
>>> print('p is {0}'.format(p))
p is (3, 4)
>>>
```

讨论¶

自定义 `__repr__()` 和 `__str__()` 通常是很好的习惯，因为它能简化调试和实例输出。 例如，如果仅仅只是打印输出或日志输出某个实例，那么程序员会看到实例更加详细与有用的信息。

`__repr__()` 生成的文本字符串标准做法是需要让 `eval(repr(x)) == x` 为真。 如果实在不能这样子做，应该创建一个有用的文本表示，并使用 < 和 > 括起来。比如：

```
>>> f = open('file.dat')
>>> f
```

```
<_io.TextIOWrapper name='file.dat' mode='r' encoding='UTF-8'>
>>>
```

如果 `__str__()` 没有被定义，那么就会使用 `__repr__()` 来代替输出。

上面的 `format()` 方法的使用看上去很有趣，格式化代码 `{0.x}` 对应的是第1个参数的x属性。 因此，在下面的函数中，0实际上指的就是 `self` 本身：

```
def __repr__(self):
    return 'Pair({0.x!r}, {0.y!r})'.format(self)
```

作为这种实现的一个替代，你也可以使用 `%` 操作符，就像下面这样：

```
def __repr__(self):
    return 'Pair(%r, %r)' % (self.x, self.y)
```