基本概念

仅仅打印“Hello World”就足够了吗？你应该想要做更多的事——你想要得到一些输入，然后做操作，再从中得到一些输出。在 Python 中，我们可以使用常量和变量来完成这些工作。

一个字面意义上的常量的例子是如同 `5`、`1.23`、`9.25e-3` 这样的数，或者如同 `'This is a string'`、`"It's a string!"`这样的字符串。它们被称作字面意义上的，因为它们具备 字面 的意义——你按照它们的字面意义使用它们的值。数 `2` 总是代表它自己，而不会是别的什么东西——它是一个常量，因为不能改变它的值。因此，所有这些都被称为字面意义上的常量。

1. 数

在 Python 中有 4 种类型的数——整数、长整数、浮点数和复数。

- `2` 是一个整数的例子。
- 长整数不过是大一些的整数。
- `3.23` 和 `52.3E-4` 是浮点数的例子。E 标记表示 10 的幂。在这里，`52.3E-4` 表示 `52.3 * 10<sup>-4</sup>`。
- `(-5+4j)`和`(2.3-4.6j)`是复数的例子。

2. 字符串

字符串是 字符的序列 。字符串基本上就是一组单词。

我几乎可以保证你在每个 Python 程序中都要用到字符串，所以请特别留心下面这部分的内容。下面告诉你如何在 Python 中使用字符串。

- **使用单引号（'）**

  你可以用单引号指示字符串，就如同`'Quote me on this'`这样。所有的空白，即空格和制表符都照原样保留。
- **使用双引号（"）**

  在双引号中的字符串与单引号中的字符串的使用完全相同，例如`"What's your name?"`。
- **使用三引号（'''或"""）**

  利用三引号，你可以指示一个多行的字符串。你可以在三引号中自由的使用单引号和双引号。例如：

```
'''This is a multi-line string. This is the first line. This is the
second line. "What's your name?," I asked. He said "Bond, James Bond."
'''
```

- **转义符**

  假设你想要在一个字符串中包含一个单引号（'），那么你该怎么指示这个字符串？例如，这个字符串是`What's your name?`。你肯定不会用`'What's your name?'`来指示它，因为 Python 会弄不明白这个字符串从何处开始，何处结束。所以，你需要指明单引号而不是字符串的结尾。可以通过 转义符 来完成这个任务。你用`\'`来指示单引号——注意这个反斜杠。现在你可以把字符串表示为`'What\'s your name?'`。
  另一个表示这个特别的字符串的方法是`"What's your name?"`，即用双引号。类似地，要在双引号字符串中使用双引号本身的时候，也可以借助于转义符。
  另外，你可以用转义符`\\`来指示反斜杠本身。
  值得注意的一件事是，在一个字符串中，行末的单独一个反斜杠表示字符串在下一行继续，而不是开始一个新的行。例如：

  ```
  "This is the first sentence.\
  This is the second sentence."
  ```

  等价于`"This is the first sentence. This is the second sentence."`

- **自然字符串**

  如果你想要指示某些不需要如转义符那样的特别处理的字符串，那么你需要指定一个自然字符串。自然字符串通过给字符串加上前缀`r`或`R`来指定。例如`r"Newlines are indicated by \n"`。

- **Unicode 字符串**

  Unicode 是书写国际文本的标准方法。如果你想要用你的母语如北印度语或阿拉伯语写文本，那么你需要有一个支持 Unicode 的编辑器。类似地，Python 允许你处理 Unicode 文本——你只需要在字符串前加上前缀`u`或`U`。例如，`u"This is a Unicode string."`。
  记住，在你处理文本文件的时候使用 Unicode 字符串，特别是当你知道这个文件含有用非英语的语言写的文本。

- **字符串是不可变的**

  这意味着一旦你创造了一个字符串，你就不能再改变它了。虽然这看起来像是一件坏事，但实际上它不是。我们将会在后面的程序中看到为什么我们说它不是一个缺点。

- **按字面意义级连字符串**

  如果你把两个字符串按字面意义相邻放着，他们会被 Python 自动级连。例如，`'What\'s' 'your name?'`会被自动转为`"What's your name?"`。
  给 C/C++程序员的注释 在 Python 中没有专门的`char`数据类型。确实没有需要有这个类型，我相信你不会为此而烦恼。

给 Perl/PHP 程序员的注释 记住，单引号和双引号字符串是完全相同的——它们没有在任何方面有不同。

给正则表达式用户的注释 一定要用自然字符串处理正则表达式。否则会需要使用很多的反斜杠。例如，后向引用符可以写成`'\\1'`或`r'\1'`。

3. 变量

仅仅使用字面意义上的常量很快就会引发烦恼——我们需要一种既可以储存信息 又可以对它们进行操作的方法。这是为什么要引入 变量 。变量就是我们想要的东西——它们的值可以变化，即你可以使用变量存储任何东西。变量只是你的计算机中存储信息的一部分内存。与字面意义上的常量不同，你需要一些能够访问这些变量的方法，因此你给变量名字。

4. 标识符的命名

变量是标识符的例子。 标识符 是用来标识 某样东西 的名字。在命名标识符的时候，你要遵循这些规则：

- 标识符的第一个字符必须是字母表中的字母（大写或小写）或者一个下划线（‘_’）。
- 标识符名称的其他部分可以由字母（大写或小写）、下划线（‘_’）或数字（0-9）组成。
- 标识符名称是对大小写敏感的。例如，`myname`和`myName`**不是**一个标识符。注意前者中的小写 n 和后者中的大写 N。
- 有效 标识符名称的例子有`i`、`__my_name`、`name_23`和`a1b2_c3`。
- 无效 标识符名称的例子有`2things`、`this is spaced out`和`my-name`。

5. 数据类型

变量可以处理不同类型的值，称为**数据类型**。基本的类型是数和字符串，我们已经讨论过它们了。在后面的章节里面，我们会研究怎么用类创造我们自己的类型。

6. 对象

记住，Python 把在程序中用到的任何东西都称为 对象 。这是从广义上说的。因此我们不会说“某某 东西 ”，我们说“某个 对象 ”。

给面向对象编程用户的注释 就每一个东西包括数、字符串甚至函数都是对象这一点来说，Python 是极其完全地面向对象的。

我们将看一下如何使用变量和字面意义上的常量。保存下面这个例子，然后运行程序。

如何编写 Python 程序 下面是保存和运行 Python 程序的标准流程。 1. 打开你最喜欢的编辑器。 2. 输入例子中的程序代码。 3. 用注释中给出的文件名把它保存为一个文件。我按照惯例把所有的 Python 程序都以扩展名`.py`保存。 4. 运行解释器命令 **python program.py** 或者使用 IDLE 运行程序。你也可以使用先前介绍的可执行的方法。

```
# Filename : var.py
i = 5
print i
i = i + 1
print i

s = '''This is a multi-line string.
This is the second line.'''
print s
```

（源文件：code/var.py）

```
$ python var.py
5
6
This is a multi-line string.
This is the second line.
```

下面来说明一下这个程序如何工作。首先我们使用赋值运算符（`=`）把一个字面意义上的常数`5`赋给变量`i`。这一行称为一个语句。语句声明需要做某件事情，在这个地方我们把变量名`i`与值`5`连接在一起。接下来，我们用`print`语句打印`i`的值，就是把变量的值打印在屏幕上。
然后我们对`i`中存储的值加`1`，再把它存回`i`。我们打印它时，得到期望的值`6`。
类似地，我们把一个字面意义上的字符串赋给变量`s`然后打印它。
给 C/C++程序员的注释 使用变量时只需要给它们赋一个值。不需要声明或定义数据类型。

7. 逻辑行与物理行

物理行是你在编写程序时所 看见 的。逻辑行是 Python 看见 的单个语句。Python 假定每个 物理行 对应一个 逻辑行 。

逻辑行的例子如`print 'Hello World'`这样的语句——如果它本身就是一行（就像你在编辑器中看到的那样），那么它也是一个物理行。

默认地，Python 希望每行都只使用一个语句，这样使得代码更加易读。

如果你想要在一个物理行中使用多于一个逻辑行，那么你需要使用分号（;）来特别地标明这种用法。分号表示一个逻辑行/语句的结束。例如：

```
i = 5
print i
```

与下面这个相同：

```
i = 5;
print i;
```

同样也可以写成：

```
i = 5; print i;
```

甚至可以写成：

```
i = 5; print i
```

然而，我**强烈建议**你坚持**在每个物理行只写一句逻辑行**。仅仅当逻辑行太长的时候，在多于一个物理行写一个逻辑行。这些都是为了尽可能避免使用分号，从而让代码更加易读。事实上，我 从来没有 在 Python 程序中使用过或看到过分号。

下面是一个在多个物理行中写一个逻辑行的例子。它被称为**明确的行连接**。

```
s = 'This is a string. \
This continues the string.'
print s
```

它的输出：

```
This is a string. This continues the string.
```

类似地，

```
print \
i
```

与如下写法效果相同：

```
print i
```

有时候，有一种暗示的假设，可以使你不需要使用反斜杠。这种情况出现在逻辑行中使用了圆括号、方括号或波形括号的时候。这被称为**暗示的行连接**。你会在后面介绍如何使用列表的章节中看到这种用法。

8. 缩进

空白在 Python 中是重要的。事实上**行首的空白是重要的**。它称为**缩进**。在逻辑行首的空白（空格和制表符）用来决定逻辑行的缩进层次，从而用来决定语句的分组。

这意味着同一层次的语句**必须**有相同的缩进。每一组这样的语句称为一个**块**。我们将在后面的章节中看到有关块的用处的例子。

你需要记住的一样东西是错误的缩进会引发错误。例如：

```
    i = 5
     print 'Value is', i # Error! Notice a single space at the start of
the line
    print 'I repeat, the value is', i
```

当你运行这个程序的时候，你会得到下面的错误：

```
      File "whitespace.py", line 4
        print 'Value is', i # Error! Notice a single space at the start of
the line
        ^
    SyntaxError: invalid syntax
```

注意，在第二行的行首有一个空格。Python 指示的这个错误告诉我们程序的语法是无效的，即程序没有正确地编写。它告诉你， 你不能随意地开始新的语句块 （当然除了你一直在使用的主块）。何时你能够使用新块，将会在后面的章节，如控制流中详细介绍。

如何缩进 **不要**混合使用制表符和空格来缩进，因为这在跨越不同的平台的时候，无法正常工作。我 强烈建议 你在每个缩进层次使用 单个制表符 或 两个或四个空格 。 选择这三种缩进风格之一。更加重要的是，选择一种风格，然后**一贯地**使用它，即 只 使用这一种风格。