3. Python 速览

下面的例子以是否显示提示符（>>> 与 ...）区分输入与输出：输入例子中的代码时，要键入以提示符开头的行中提示符后的所有内容；未以提示符开头的行是解释器的输出。注意，例子中的某行出现的第二个提示符是用来结束多行命令的，此时，要键入一个空白行。

你可以通过在示例方块右上角的 >>> 上点击来切换显示提示符和输出。 如果你隐藏了一个示例的提示符和输出，那么你可以方便地将输入行复制并粘贴到你的解释器中。

本手册中的许多例子，甚至交互式命令都包含注释。Python 注释以 # 开头，直到该物理行结束。注释可以在行开头，或空白符与代码之后，但不能在字符串里面。字符串中的 # 号就是 # 号。注释用于阐明代码，Python 不解释注释，键入例子时，可以不输入注释。

示例如下：

```
# this is the first comment
spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

3.1. Python 用作计算器

现在，尝试一些简单的 Python 命令。启动解释器，等待主提示符（>>> ）出现。

3.1.1. 数字

解释器像一个简单的计算器：输入表达式，就会给出答案。表达式的语法很直接：运算符 +、-、*、/ 的用法和其他大部分语言一样（比如，Pascal 或 C）；括号 (()) 用来分组。例如：

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating point number
1.6
```

整数（如，2、4、20 ）的类型是 int，带小数（如，5.0、1.6 ）的类型是 float。本教程后半部分将介绍更多数字类型。

除法运算（/）返回浮点数。用 // 运算符执行 floor division 的结果是整数（忽略小数）；计算余数用 %：

```
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3  # floor division discards the fractional part
5
>>> 17 % 3  # the % operator returns the remainder of the division
2
>>> 5 * 3 + 2  # floored quotient * divisor + remainder
17
```

Python 用 ** 运算符计算乘方 [1]：

```
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

等号（=）用于给变量赋值。赋值后，下一个交互提示符的位置不显示任何结果：

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

如果变量未定义（即，未赋值），使用该变量会提示错误：

```
>>> n  # try to access an undefined variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Python 全面支持浮点数；混合类型运算数的运算会把整数转换为浮点数：

```
>>> 4 * 3.75 - 1
14.0
```

交互模式下，上次输出的表达式会赋给变量 `_`。把 Python 当作计算器时，用该变量实现下一步计算更简单，例如：

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

最好把该变量当作只读类型。不要为它显式赋值，否则会创建一个同名独立局部变量，该变量会用它的魔法行为屏蔽内置变量。

除了 `int` 和 `float`，Python 还支持其他数字类型，例如 `Decimal` 或 `Fraction`。Python 还内置支持 复数，后缀 `j` 或 `J` 用于表示虚数（例如 `3+5j` ）。

3.1.2. 字符串

除了数字，Python 还可以操作字符串。字符串有多种表现形式，用单引号（`'……'`）或双引号（`"……"`）标注的结果相同 [2]。反斜杠 `\` 用于转义：

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

交互式解释器会为输出的字符串加注引号，特殊字符使用反斜杠转义。虽然，有时输出的字符串看起来与输入的字符串不一样（外加的引号可能会改变），但两个字符串是相同的。如果字符串中有单引号而没有双引号，该字符串外将加注双引号，反之，则加注单引号。`print()` 函数输出的内容更简洁易读，它会省略两边的引号，并输出转义后的特殊字符：

```
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
>>> print('"Isn\'t," they said.')
"Isn't," they said.
>>> s = 'First line.\nSecond line.'  # \n means newline
>>> s  # without print(), \n is included in the output
'First line.\nSecond line.'
>>> print(s)  # with print(), \n produces a new line
First line.
Second line.
```

如果不希望前置 `\` 的字符转义成特殊字符，可以使用 *原始字符串*，在引号前添加 `r` 即可：

```
>>> print('C:\some\name')  # here \n means newline!
C:\some
ame
>>> print(r'C:\some\name')  # note the r before the quote
C:\some\name
```

字符串字面值可以包含多行。 一种实现方式是使用三重引号：`"""..."""` 或 `'''...'''`。 字符串中将自动包括行结束符，但也可以在换行的地方添加一个 `\` 来避免此情况。 参见以下示例：

```
print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
```

输出如下（请注意开始的换行符没有被包括在内）：

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

字符串可以用 `+` 合并（粘到一起），也可以用 `*` 重复：

```
>>> # 3 times 'un', followed by 'ium'
>>> 3 * 'un' + 'ium'
'ununium'
```

相邻的两个或多个 *字符串字面值*（引号标注的字符）会自动合并：

```
>>> 'Py' 'thon'
'Python'
```

拼接分隔开的长字符串时，这个功能特别实用：

```
>>> text = ('Put several strings within parentheses '
...         'to have them joined together.')
>>> text
'Put several strings within parentheses to have them joined together.'
```

这项功能只能用于两个字面值，不能用于变量或表达式：

```
>>> prefix = 'Py'
>>> prefix 'thon'  # can't concatenate a variable and a string literal
  File "<stdin>", line 1
    prefix 'thon'
           ^
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
  File "<stdin>", line 1
    ('un' * 3) 'ium'
               ^
SyntaxError: invalid syntax
```

合并多个变量，或合并变量与字面值，要用 `+`：

```
>>> prefix + 'thon'
'Python'
```

字符串支持 *索引*（下标访问），第一个字符的索引是 0。单字符没有专用的类型，就是长度为一的字符串：

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
```

索引还支持负数，用负数索引时，从右边开始计数：

```
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'
```

注意，-0 和 0 一样，因此，负数索引从 -1 开始。

除了索引，字符串还支持 *切片*。索引可以提取单个字符，*切片* 则提取子字符串：

```
>>> word[0:2]  # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5 (excluded)
'tho'
```

切片索引的默认值很有用；省略开始索引时，默认值为 0，省略结束索引时，默认为到字符串的结尾：

```
>>> word[:2]   # character from the beginning to position 2 (excluded)
'Py'
>>> word[4:]   # characters from position 4 (included) to the end
'on'
>>> word[-2:]  # characters from the second-last (included) to the end
'on'
```

注意，输出结果包含切片开始，但不包含切片结束。因此，`s[:i] + s[i:]` 总是等于 `s`：

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

还可以这样理解切片，索引指向的是字符 *之间*，第一个字符的左侧标为 0，最后一个字符的右侧标为 *n*，*n* 是字符串长度。例如：

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

第一行数字是字符串中索引 0...6 的位置，第二行数字是对应的负数索引位置。*i* 到 *j* 的切片由 *i* 和 *j* 之间所有对应的字符组成。

对于使用非负索引的切片，如果两个索引都不越界，切片长度就是起止索引之差。例如，`word[1:3]` 的长度是 2。

索引越界会报错：

```
>>> word[42]  # the word only has 6 characters
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

但是，切片会自动处理越界索引：

```
>>> word[4:42]
'on'
>>> word[42:]
''
```

Python 字符串不能修改，是 immutable 的。因此，为字符串中某个索引位置赋值会报错：

```
>>> word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

要生成不同的字符串，应新建一个字符串：

```
>>> 'J' + word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```

内置函数 `len()` 返回字符串的长度：

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

参见：

文本序列类型 --- str
字符串是 *序列类型*，支持序列类型的各种操作。

字符串的方法
字符串支持很多变形与查找方法。

格式字符串字面值
内嵌表达式的字符串字面值。

格式字符串语法
使用 `str.format()` 格式化字符串。

printf 风格的字符串格式化
这里详述了用 `%` 运算符格式化字符串的操作。

3.1.3. 列表

Python 支持多种 *复合* 数据类型，可将不同值组合在一起。最常用的 *列表*，是用方括号标注，逗号分隔的一组值。*列表* 可以包含不同类型的元素，但一般情况下，各个元素的类型相同：

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

和字符串（及其他内置 sequence 类型）一样，列表也支持索引和切片：

```
>>> squares[0]  # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]
```

切片操作返回包含请求元素的新列表。以下切片操作会返回列表的 浅拷贝：

```
>>> squares[:]
[1, 4, 9, 16, 25]
```

列表还支持合并操作：

```
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

与 immutable 字符串不同, 列表是 mutable 类型，其内容可以改变：

```
>>> cubes = [1, 8, 27, 65, 125]  # something's wrong here
>>> 4 ** 3  # the cube of 4 is 64, not 65!
64
>>> cubes[3] = 64  # replace the wrong value
>>> cubes
[1, 8, 27, 64, 125]
```

`append()` *方法* 可以在列表结尾添加新元素（详见后文）：

```
>>> cubes.append(216)  # add the cube of 6
>>> cubes.append(7 ** 3)  # and the cube of 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

为切片赋值可以改变列表大小，甚至清空整个列表：

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # clear the list by replacing all the elements with an empty list
>>> letters[:] = []
>>> letters
[]
```

内置函数 `len()` 也支持列表：

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

还可以嵌套列表（创建包含其他列表的列表），例如：

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

3.2. 走向编程的第一步

当然，Python 还可以完成比二加二更复杂的任务。 例如，可以编写 斐波那契数列 的初始子序列，如下所示：

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while a < 10:
...     print(a)
...     a, b = b, a+b
...
0
1
1
2
3
5
8
```

本例引入了几个新功能。

- 第一行中的 *多重赋值*：变量 `a` 和 `b` 同时获得新值 0 和 1。最后一行又用了一次多重赋值，这体现在右表达式在赋值前就已经求值了。右表达式求值顺序为从左到右。
- `while` 循环只要条件（这里指：`a < 10`）保持为真就会一直执行。Python 和 C 一样，任何非零整数都为真，零为假。这个条件也可以是字符串或列表的值，事实上，任何序列都可以；长度非零就为真，空序列则为假。示例中的判断只是最简单的比较。比较操作符的标准写法和 C 语言一样：`<`（小于）、`>`（大于）、`==`（等于）、`<=`（小于等于)、`>=`（大于等于）及 `!=`（不等于）。
- *循环体* 是 *缩进的*：缩进是 Python 组织语句的方式。在交互式命令行里，得为每个缩输入制表符或空格。使用文本编辑器可以实现更复杂的输入方式；所有像样的文本编辑器都支持自动缩进。交互式输入复合语句时, 要在最后输入空白行表示结束（因为解析器不知道哪一行代码是最后一行）。注意，同一块语句的每一行的缩进相同。
- `print()` 函数输出给定参数的值。与表达式不同（比如，之前计算器的例子），它能处理多个参数，包括浮点数与字符串。它输出的字符串不带引号，且各参数项之间会插入一个空格，这样可以实现更好的格式化操作：

  ```
  >>> i = 256*256
  >>> print('The value of i is', i)
  The value of i is 65536
  ```

  关键字参数 *end* 可以取消输出后面的换行, 或用另一个字符串结尾：

```
>>> a, b = 0, 1
>>> while a < 1000:
...     print(a, end=',')
...     a, b = b, a+b
...
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```

备注

[1] `**` 比 `-` 的优先级更高, 所以 `-3**2` 会被解释成 `-(3**2)` ，因此，结果是 `-9`。要避免这个问题，并且得到 `9`, 可以用 `(-3)**2`。

[2] 和其他语言不一样，特殊字符如 `\n` 在单引号（`'...'`）和双引号（`"..."`）里的意义一样。这两种引号唯一的区别是，不需要在单引号里转义双引号 `"`，但必须把单引号转义成 `\'`，反之亦然。

©版权所有 2001-2022, Python Software Foundation.
This page is licensed under the Python Software Foundation License Version 2.
Examples, recipes, and other code in the documentation are additionally licensed under the Zero Clause BSD License.
See History and License for more information.

The Python Software Foundation is a non-profit corporation. Please donate.

最后更新于 9月 15, 2022. Found a bug?
Created using Sphinx 3.4.3.