没有单独的 `long` 类型。`int` 类型可以指任何大小的整数。

你可以通过使用三个引号——`"""` 或 `'''` 来指定多行字符串。你可以在三引号之间自由地使用单引号与双引号。[illegible]

字符串是不可变的

格式化方法

有时候我们会想要从其他信息中构建字符串。这正是 `format()` 方法大有用武之地的地方。

将以下内容保存为文件 `str_format.py` ：

```
age = 20
name = 'Swaroop'

print('{0} was {1} years old when he wrote this book'.format(name, age))
print('Why is {0} playing with that python?'.format(name))
```

输出：

```
$ python str_format.py
Swaroop was 20 years old when he wrote this book
Why is Swaroop playing with that python?
```

```
# 对于浮点数 '0.333' 保留小数点(.)后三位
print('{0:.3f}'.format(1.0/3))
# 使用下划线填充文本，并保持文字处于中间位置
# 使用 (^) 定义 '___hello___'字符串长度为 11
print('{0:_^11}'.format('hello'))
# 基于关键词输出 'Swaroop wrote A Byte of Python'
print('{name} wrote {book}'.format(name='Swaroop', book='A Byte of Python'))
```

输出：

```
0.333
___hello___
Swaroop wrote A Byte of Python
```

由于我们正在讨论格式问题，就要注意 `print` 总是会以一个不可见的“新一行”字符（`\n`）结尾，因此重复调用 `print` 将会在相互独立的一行中分别打印。为防止打印过程中出现这一换行符，你可以通过 `end` 指定其应以空白结尾：

```
print('a', end='')
print('b', end='')
```

还有一件需要的事情，在一个字符串中，一个放置在末尾的反斜杠表示字符串将在下一行继续，但不会添加新的一行。来看看例子：

原始字符串

如果你需要指定一些未经过特殊处理的字符串，比如转义序列，那么你需要在字符串前增加 `r` 或 `R` 来指定一个 原始（*Raw*） 字符串[4]。下面是一个例子：

```
r"Newlines are indicated by \n"
```

> 针对正则表达式用户的提示
>
> 在处理正则表达式时应全程使用原始字符串。否则，将会有大量 Backwhacking 需要处理。举例说明的话，反向引用可以通过 `'\\1'` 或 `r'\1'` 来实现。

Python 是强（Strongly）面向对象的，因为所有的一切都是对象， 包括数字、字符串与函数。

- `13 / 3` 输出 `4.333333333333333`。
- `//` （整除）
  - x 除以 y 并对结果向下取整至最接近的整数。
  - `13 // 3` 输出 `4`。
  - `-13 // 3` 输出 `-5`。
- `~` （按位取反）[4]
  - x 的按位取反结果为 -(x+1)。

- `<` （小于）
  - 返回 x 是否小于 y。所有的比较运算符返回的结果均为 `True` 或 `False`。请注意这些名称之中的大写字母。
  - `5 < 3` 输出 `False`，`3 < 6` 输出 `True`。
  - 比较可以任意组成组成链接：`3 < 5 < 7` 返回 `True`。

- `not`（布尔“非”）[5]
  - 如果 x 是 `Ture`，则返回 `False`。如果 x 是 `False`，则返回 `True`。
  - `x = Ture; not x` 返回 `False`。
- `and`（布尔“与”）[6]
  - 如果 x 是 `False`，则 `x and y` 返回 `False`，否则返回 y 的计算值。
  - 当 x 是 `False` 时，`x = False; y = True; x and y` 将返回 `False`。在这一情境中，Python 将不会计算 y，因为它已经了解 and 表达式的左侧是 `False`，这意味着整个表达式都将是 `False` 而不会是别的值。这种情况被称作短路计算（Short-circuit Evaluation）。
- `or`（布尔“或”）[7]
  - 如果 x 是 `True`，则返回 `True`，否则它将返回 y 的计算值。
  - `x = Ture; y = False; x or y` 将返回 `Ture`。在这里短路计算同样适用。

`lambda`：Lambda 表达式

`if - else`：条件表达式

`or`：布尔“或”

`and`：布尔“与”

`not x`：布尔“非”

`in, not in, is, is not, <, <=, >, >=, !=, ==`：比较，包括成员资格测试（Membership Tests）和身份测试（Identity Tests）。

`|`：按位或

`^`：按位异或

`&`：按位与

`<<, >>`：移动

`+, -`：加与减

`*, /, //, %`：乘、除、整除、取余

`+x, -x, ~x`：正、负、按位取反

`**`：求幂

```
number = 23
guess = int(input('Enter an integer : '))

if guess == number:
    # 新块从这里开始
    print('Congratulations, you guessed it.')
    print('(but you do not win any prizes!)')
    # 新块在这里结束
elif guess < number:
    # 另一代码块
    print('No, it is a little higher than that')
    # 你可以在此做任何你希望在该代码块内进行的事情
else:
    print('No, it is a little lower than that')
    # 你必须通过猜测一个大于（>）设置数的数字来到达这里。
```

Python 中不存在 `switch` 语句。

`for...in` 语句是另一种循环语句，其特点是会在一系列对象上进行迭代（*Iterates*），意即它会遍历序列中的每一个项目。[illegible]在你所需要的就是所谓队列就是一系列项目的有序集合。

案例（保存为 `for.py`）：

```
for i in range(1, 5):
    print(i)
else:
    print('The for loop is over')
```

另外需要注意的是，`range()` 每次只会生成一个数字，如果你希望获得完整的数字列表，要在使用 `range()` 时调用 `list()`。例如下面这样：`list(range(5))`，它将会返回 `[0, 1, 2, 3, 4]`。[illegible]

```
print('Length of the string is', len(s))
```

长度

函数可以通过关键字 `def` 来定义。这一关键字后跟一个函数的标识符名称，再跟一对圆括号，其中可以包括一些变量的名称，再以冒号结尾，结束这一行。随后而来的语句块是函数的一部分。下面的案例将会展示出这其实非常简单：

案例（保存为 `functon1.py` ）：

```
def say_hello():
    # 该块属于这一函数
    print('hello world')
# 函数结束

say_hello()  # 调用函数
say_hello()  # 再次调用函数
```

global 语句

全局变量

```
def say(message, times=1):
    print(message * times)

say('Hello')
say('World', 5)
```

默认参数

输出：

```
$ python function_default.py
Hello
WorldWorldWorldWorldWorld
```

要注意到如果 `return` 语句没有搭配任何一个值则代表着 `返回 None` 。`None` 在 Python 中一个特殊的类型，代表着虚无。举个例子，它用于指示一个变量没有值，如果有值则它的值便是 `None（虚无）` 。

每一个函数都在其末尾隐含了一句 `return None` ，除非你写了你自己的 `return` 语句

DocStrings

Python 有一个甚是优美的功能称作文档字符串（*Documentation Strings*），在称呼它时通常会使用另一个短一些的名字*docstrings*。DocStrings 是一款你应当使用的重要工具，它能够帮助你更好地记录程序并让其更加易于理解。令人惊叹的是，当程序实际运行时，我们甚至可以通过一个函数来获取文档！

编写你自己的模块

编写你自己的模块很简单，这其实就是你一直在做的事情！这是因为每一个 Python 程序同时也是一个模块。你只需要保证它以 `.py` 为扩展名即可。下面的案例会作出清晰的解释。

案例（保存为 `mymodule.py`）：

```
def say_hi():
    print('Hi, this is mymodule speaking.')

__version__ = '0.1'
```

上方所呈现的就是一个简单的模块。正如你所看见的，与我们一般所使用的 Python 的程序相比其实并没有什么特殊的区别。我们接下来将看到如何在其它 Python 程序中使用这一模块。

要记住该模块应该放置于与其它我们即将导入这一模块的程序相同的目录下，或者是放置在 `sys.path` 所列出的其中一个目录下。

另一个模块（保存为 `mymodule_demo.py`）：

```
import mymodule

mymodule.say_hi()
print('Version', mymodule.__version__)
```

输出：

```
$ python mymodule_demo.py
Hi, this is mymodule speaking.
Version 0.1
```

内置的 `dir()` 函数能够返回由对象所定义的名称列表。如果这一对象是一个模块，则该列表会包括函数内所定义的函数、类与变量。

该函数接受参数。如果参数是模块名称，函数将返回这一指定模块的名称列表。如果没有提供参数，函数将返回当前模块的名称列表。

Python 中有四种内置的数据结构——列表（*List*）、元组（*Tuple*）、字典（*Dictionary*）和集合（*Set*）。

```
shoplist = ['apple', 'mango', 'carrot', 'banana']

print('I have', len(shoplist), 'items to purchase.')

print('These items are:', end=' ')
for item in shoplist:
    print(item, end=' ')
```

如果你想了解列表对象定义的所有方法，可以通过 `help(list)` 来了解更多细节。

元组（Tuple）用于将多个对象保存到一起。你可以将它们近似地看作列表，但是元组不能提供列表类能够提供给你的广泛的功能。元组的一大特征类似于字符串，它们是不可变的，也就是说，你不能编辑或更改元组。

元组是通过特别指定项目来定义的，在指定项目时，你可以给它们加上括号，并在括号内部用逗号进行分隔。

元组通常用于保证某一语句或某一用户定义的函数可以安全地采用一组数值，意即元组内的数值不会改变。

```
zoo = ('python', 'elephant', 'penguin')
```

```
zoo = ('python', 'elephant', 'penguin')
print('Number of animals in the zoo is', len(zoo))

new_zoo = 'monkey', 'camel', zoo
print('Number of cages in the new zoo is', len(new_zoo))
print('All animals in new zoo are', new_zoo)
print('Animals brought from old zoo are', new_zoo[2])
print('Last animal brought from old zoo is', new_zoo[2][2])
print('Number of animals in the new zoo is',
      len(new_zoo)-1+len(new_zoo[2]))
```

相当于二维

出：

元组其实类似序列

据结构

```
$ python ds_using_tuple.py
Number of animals in the zoo is 3
Number of cages in the new zoo is 3
All animals in new zoo are ('monkey', 'camel', ('python', 'elephant', 'penguin'))
Animals brought from old zoo are ('python', 'elephant', 'penguin')
Last animal brought from old zoo is penguin
Number of animals in the new zoo is 5
```

每一个 Python 模块都定义了它的 `__name__` 属性。如果它与 `__main__` 属性相同则代表这一模块是由用户独立运行的，因此我们便可以采取适当的行动。

这是因为每一个 Python 程序同时也是一个模块。你只需要保证它以 `.py` 为扩展名即可。

案例（保存为 `mymodule.py`）：

```
def say_hi():
    print('Hi, this is mymodule speaking.')

__version__ = '0.1'
```

任意.py文件如name.py都是其他py文件的模块
name.py中定义的函数可被其他代码重用

上方所呈现的就是一个简单的模块。正如你所看见的，与我们一般所使用的 Python 的程序相比其实并没有什么特殊的区别。我们接下来将看到如何在其它 Python 程序中使用这一模块。

要记住该模块应该放置于与其它我们即将导入这一模块的程序相同的目录下，或者是放置在 `sys.path` 所列出的其中一个目录下 其实就相当于c/c++中的自己编写的头文件

另一个模块（保存为 `mymodule_demo.py`）：

```
import mymodule

mymodule.say_hi()
print('Version', mymodule.__version__)
```

输出：

```
$ python mymodule_demo.py
Hi, this is mymodule speaking.
Version 0.1
```

```
>>> dir(sys)
['__displayhook__', '__doc__',
'argv', 'builtin_module_names',
'version', 'version_info']
```

运行 `dir(str)` 可以访问 `str`（String，字符

包是指一个包含模块与一个特殊的 `__init__.py` 文件的文件夹，后者向 Python 表明这一文件夹是特别的，因为其包含了 Python 模块。

建设你想创建一个名为“world”的包，其中还包含着 ”asia“、”africa“等其它子包，同时这些子包都包含了诸如”india“、”madagascar“等模块。

下面是你会构建出的文件夹的结构：

```
- <some folder present in the sys.path>/
    - world/
        - __init__.py
        - asia/
            - __init__.py
            - india/
                - __init__.py
                - foo.py
        - africa/
            - __init__.py
            - madagascar/
                - __init__.py
                - bar.py
```

每个子包都含有__init__.py 文件

Python 中有四种内置的数据结构——列表（*List*）、元组（*Tuple*）、字典（*Dictionary*）和集合（*Set*）

包含 **0** 或 **1** 个项目的元组

可用tuplename[n]调用元素

一个空的元组由一对圆括号构成，就像 `myempty = ()` 这样。然而，一个只拥有一个项目的元组并不像这样简单。你必须在第一个（也是唯一一个）项目的后面加上一个逗号来指定它，如此一来 Python 才可以识别出在这个表达式想表达的究竟是一个元组还是只是一个被括号所环绕的对象，也就是说，如果你想指定一个包含项目 `2` 的元组，你必须指定 `singleton = (2, )`。

```
print('Item 1 to 3 is', shoplist[1:3])
print('Item 2 to end is', shoplist[2:])
print('Item 1 to -1 is', shoplist[1:-1])
print('Item start to end is', shoplist[:])
```

指定特殊的元素，

为空，说明全部是

```
bri = set(['brazil', 'russia', 'india'])
```

集合

```python
name = 'Swaroop'

if name.startswith('Swa'):
    print('Yes, the string starts with "Swa"')

if 'a' in name:
    print('Yes, it contains the string "a"')

if name.find('war') != -1:
    print('Yes, it contains the string "war"')

delimiter = '_*_'
mylist = ['Brazil', 'Russia', 'India', 'China']
print(delimiter.join(mylist))
```

类与对象是面向对象编程的两个主要方面。一个类（**Class**）能够创建一种新的类型（*Type*），其中对象（**Object**）就是类的实例（**Instance**）。可以这样来类比：你可以拥有类型 `int` 的变量，也就是说存储整数的变量是 `int` 类的实例（对象）。

匹配一个范围[ ]：[A-Z0-9]

匹配前面出现的正则表达式任意多次，包含0次 * ：[abc]*

匹配前面出现的正则表达式0次或一次?： .*?

匹配前面出现的正则表达式一次次或多次+：a+

\d：匹配任何数字

\s：任何空白符 \n\t\r\v\f

\w：匹配任何数字、字母、字符 == [A-Za-z0-9_]

*符号如果在范围中[ ]出现，那么他代表的意思为否定

比如[^a]，那么就是匹配除了a之外的任意一个字符。

组件组，对关键数据进行获取()

<a href=(.*?)> 对路由地址进行获取

```
In [14]: def total(a=5, *numbers, **phonebook):
             print('a', a)

             for single_item in numbers:
                 print('single_item',single_item)

             for first_part, second_part in phonebook.items():
                 print(first_part,second_part)
         print(total(10,1,2,3,Jack=1123,John=2231,Inge=1560))
```

```
a 10
single_item 1
```

2

```
single_item 2
single_item 3
Jack 1123
John 2231
Inge 1560
None
```

```
def x(b,*m,**n):
    print(b)
    for i in m:
        print(i)
    for i in n:
        print(i)
    for i ,j in n.items():# 分别输出第一个第二个，但不是元组,且非字符串：x 2
        print(i,j)
    for i in n.items():#输出元组，且将第一个视为字符串('x', 2)
        print(i)
print(x(3,3,4,5,6,7,x=2,y=3,z=999))
```

```
In [2]: import time
        thisyear=int(time.strftime("%Y", time.localtime()))
        print(thisyear)
```

```
2018
```

```
In [5]: print (time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))
```

```
2018-01-23 10:25:01
```