**读取键盘输入**

Python提供了 input() 内置函数从标准输入读入一行文本，默认的标准输入是键盘。

input 可以接收一个Python表达式作为输入，并将运算结果返回。

#!/usr/bin/python3

str = input("请输入：");

print ("你输入的内容是: ", str)

这会产生如下的对应着输入的结果：

请输入：菜鸟教程

你输入的内容是: 菜鸟教程

**读和写文件**

open() 将会返回一个 file 对象，基本语法格式如下:

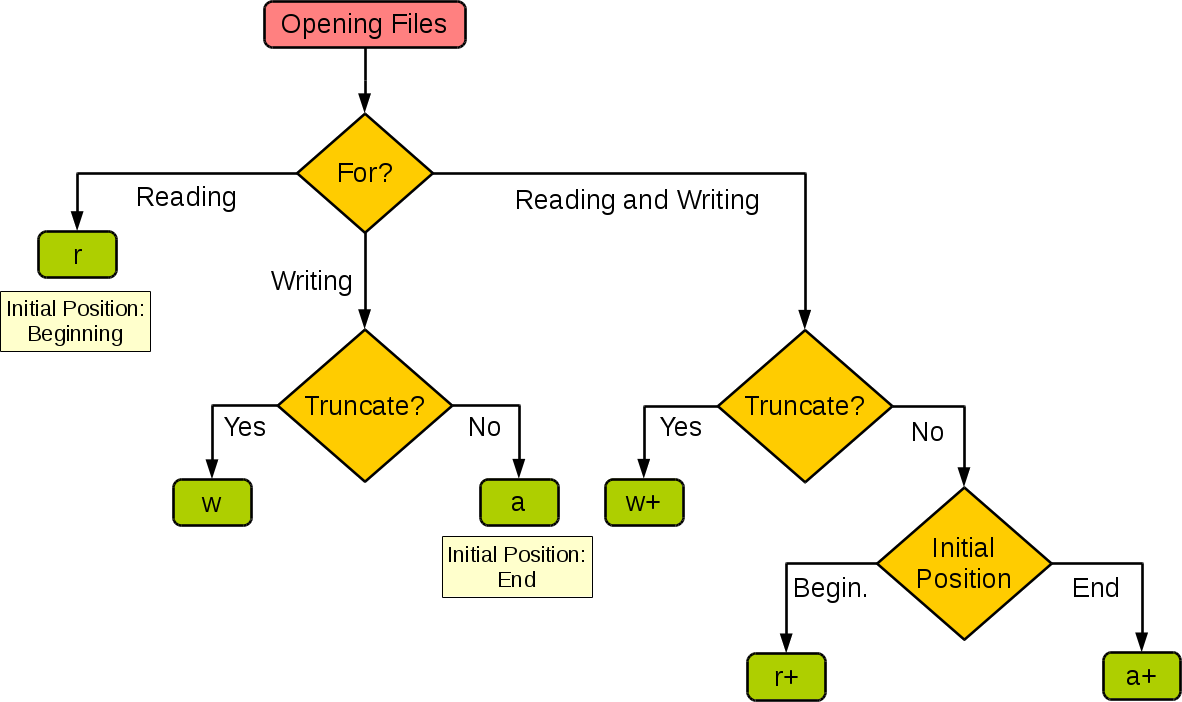
open(filename, mode)

* filename：包含了你要访问的文件名称的字符串值。
* mode：决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。

不同模式打开文件的完全列表：

|  |  |
| --- | --- |
| **模式** | **描述** |
| r | 以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。 |
| rb | 以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。 |
| r+ | 打开一个文件用于读写。文件指针将会放在文件的开头。 |
| rb+ | 以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。 |
| w | 打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| wb | 以二进制格式打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| w+ | 打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| wb+ | 以二进制格式打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 |
| a | 打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| ab | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 |
| a+ | 打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。 |
| ab+ | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。 |

下图很好的总结了这几种模式：



| **模式** | **r** | **r+** | **w** | **w+** | **a** | **a+** |
| --- | --- | --- | --- | --- | --- | --- |
| 读 | + | + |  | + |  | + |
| 写 |  | + | + | + | + | + |
| 创建 |  |  | + | + | + | + |
| 覆盖 |  |  | + | + |  |  |
| 指针在开始 | + | + | + | + |  |  |
| 指针在结尾 |  |  |  |  | + | + |

以下实例将字符串写入到文件 foo.txt 中：

#!/usr/bin/python3

# 打开一个文件

f = open("/tmp/foo.txt", "w")

f.write( "Python 是一个非常好的语言。\n是的，的确非常好!!\n" )

# 关闭打开的文件

f.close()

* 第一个参数为要打开的文件名。
* 第二个参数描述文件如何使用的字符。 mode 可以是 'r' 如果文件只读, 'w' 只用于写 (如果存在同名文件则将被删除), 和 'a' 用于追加文件内容; 所写的任何数据都会被自动增加到末尾. 'r+' 同时用于读写。 mode 参数是可选的; 'r' 将是默认值。

此时打开文件 foo.txt,显示如下：

$ cat /tmp/foo.txt

Python 是一个非常好的语言。

是的，的确非常好!!

**文件对象的方法**

本节中剩下的例子假设已经创建了一个称为 f 的文件对象。

**f.read()**

为了读取一个文件的内容，调用 f.read(size), 这将读取一定数目的数据, 然后作为字符串或字节对象返回。

size 是一个可选的数字类型的参数。 当 size 被忽略了或者为负, 那么该文件的所有内容都将被读取并且返回。

以下实例假定文件 foo.txt 已存在（上面实例中已创建）：

#!/usr/bin/python3

# 打开一个文件

f = open("/tmp/foo.txt", "r")

str = f.read()

print(str)

# 关闭打开的文件

f.close()

执行以上程序，输出结果为：

Python 是一个非常好的语言。

是的，的确非常好!!

**f.readline()**

f.readline() 会从文件中读取单独的一行。换行符为 '\n'。f.readline() 如果返回一个空字符串, 说明已经已经读取到最后一行。

#!/usr/bin/python3

# 打开一个文件

f = open("/tmp/foo.txt", "r")

str = f.readline()

print(str)

# 关闭打开的文件

f.close()

执行以上程序，输出结果为：

Python 是一个非常好的语言。

**f.readlines()**

f.readlines() 将返回该文件中包含的所有行。

如果设置可选参数 sizehint, 则读取指定长度的字节, 并且将这些字节按行分割。

#!/usr/bin/python3

# 打开一个文件

f = open("/tmp/foo.txt", "r")

str = f.readlines()

print(str)

# 关闭打开的文件

f.close()

执行以上程序，输出结果为：

['Python 是一个非常好的语言。\n', '是的，的确非常好!!\n']

另一种方式是迭代一个文件对象然后读取每行:

#!/usr/bin/python3

# 打开一个文件

f = open("/tmp/foo.txt", "r")

for line in f:

print(line, end='')

# 关闭打开的文件

f.close()

执行以上程序，输出结果为：

Python 是一个非常好的语言。

是的，的确非常好!!

这个方法很简单, 但是并没有提供一个很好的控制。 因为两者的处理机制不同, 最好不要混用。

**f.write()**

f.write(string) 将 string 写入到文件中, 然后返回写入的字符数。

#!/usr/bin/python3

# 打开一个文件

f = open("/tmp/foo.txt", "w")

num = f.write( "Python 是一个非常好的语言。\n是的，的确非常好!!\n" )

print(num)

# 关闭打开的文件

f.close()

执行以上程序，输出结果为：

29

如果要写入一些不是字符串的东西, 那么将需要先进行转换:

#!/usr/bin/python3

# 打开一个文件

f = open("/tmp/foo1.txt", "w")

value = ('www.runoob.com', 14)

s = str(value)

f.write(s)

# 关闭打开的文件

f.close()

执行以上程序，打开 foo1.txt 文件：

$ cat /tmp/foo1.txt

('www.runoob.com', 14)

**f.tell()**

f.tell() 返回文件对象当前所处的位置, 它是从文件开头开始算起的字节数。

**f.seek()**

如果要改变文件当前的位置, 可以使用 f.seek(offset, from\_what) 函数。

from\_what 的值, 如果是 0 表示开头, 如果是 1 表示当前位置, 2 表示文件的结尾，例如：

* seek(x,0) ： 从起始位置即文件首行首字符开始移动 x 个字符
* seek(x,1) ： 表示从当前位置往后移动x个字符
* seek(-x,2)：表示从文件的结尾往前移动x个字符

from\_what 值为默认为0，即文件开头。下面给出一个完整的例子：

>>> f = open('/tmp/foo.txt', 'rb+')

>>> f.write(b'0123456789abcdef')

16

>>> f.seek(5) # 移动到文件的第六个字节

5

>>> f.read(1)

b'5'

>>> f.seek(-3, 2) # 移动到文件的倒数第三字节

13

>>> f.read(1)

b'd'

**f.close()**

在文本文件中 (那些打开文件的模式下没有 b 的), 只会相对于文件起始位置进行定位。

当你处理完一个文件后, 调用 f.close() 来关闭文件并释放系统的资源，如果尝试再调用该文件，则会抛出异常。

>>> f.close()

>>> f.read()

Traceback (most recent call last):

File "<stdin>", line 1, in ?

ValueError: I/O operation on closed file

当处理一个文件对象时, 使用 with 关键字是非常好的方式。在结束后, 它会帮你正确的关闭文件。 而且写起来也比 try - finally 语句块要简短:

>>> with open('/tmp/foo.txt', 'r') as f:

... read\_data = f.read()

>>> f.closed

True

文件对象还有其他方法, 如 isatty() 和 trucate(), 但这些通常比较少用。