

文件和异常

- [文件和异常](#)
 - [1、从文件中读取数据](#)
 - [1.1 读取整个文件](#)
 - [1.2 按行读取](#)
 - [1.3 创建一个包含文件各行内容的列表](#)
 - [1.4 一些操作](#)
 - [2、写入文件](#)
 - [3、异常](#)
 - [3.1 异常使用](#)
 - [3.2 使用多个文件](#)
 - [3.3 失败时一声不吭](#)
 - [4、存储数据](#)
 - [4.1 使用 json.dump\(\)和 json.load\(\)](#)
 - [4.2 json.dumps\(\)和 json.loads\(\)](#)

1、从文件中读取数据

要使用文本文件中的信息，首先需要将信息读取到内存中。为此，你可以一次性读取文件的全部内容，也可以以每次一行的方式逐步读取。

1.1 读取整个文件

```
with open('处理所需文件/pi_digits.txt') as file_object:
    contents = file_object.read()
    print(contents.rstrip())
```

- 函数 `open()` :
接受一个参数要打开的文件的名称, `open('pi_digits.txt')` 返回一个表示文件 `pi_digits.txt` 的对象
- 关键字 `with` :
在不再需要访问文件后将其关闭,如果主动去 `close` 文件, 有可能出现很多状况。所以让 Python 去确定: 你只管打开文件, 并在需要时使用它, Python 自会在合适的时候自动将其关闭。

1.2 按行读取

读取文件时, 常常需要检查其中的每一行: 你可能要在文件中查找特定的信息, 或者要以某种方式修改文件中的文本。例如, 你可能要遍历一个包含天气数据的文件, 并使用天气描述中包含字样 `sunny` 的行。

使用 `for` 来遍历每一行

```
with open('处理所需文件/pi_digits.txt') as file_object:
    for line in file_object:
        # 会多一个空行
```

```
# print(line)
print(line.rstrip())
```

当我们使用 `print` 的时候，会多一个空行，是因为，文件每一行结束的时候有一个换行符，`print` 打印的时候会带一个换行导致多换一行

```
3.1415926535
```

```
8979323846
```

```
2643383279
```

1.3 创建一个包含文件各行内容的列表

使用关键字 `with` 时，`open()` 返回的文件对象只在 `with` 代码块内可用。如果要在 `with` 代码块外访问文件的内容，可在 `with` 代码块内将文件的各行存储在一个列表中，并在 `with` 代码块外使用该列表

```
with open('处理所需文件/pi_digits.txt') as file_object:
    lines = file_object.readlines()
    for line in lines:
        print(line.rstrip())
```

注意:

读取文本文件时，Python 将其中的所有文本都解读为字符串。如果你读取的是数字，并要将其作为数值使用，就必须使用函数 `int()` 将其转换为整数，或使用函数 `float()` 将其转换为浮点数。

1.4 一些操作

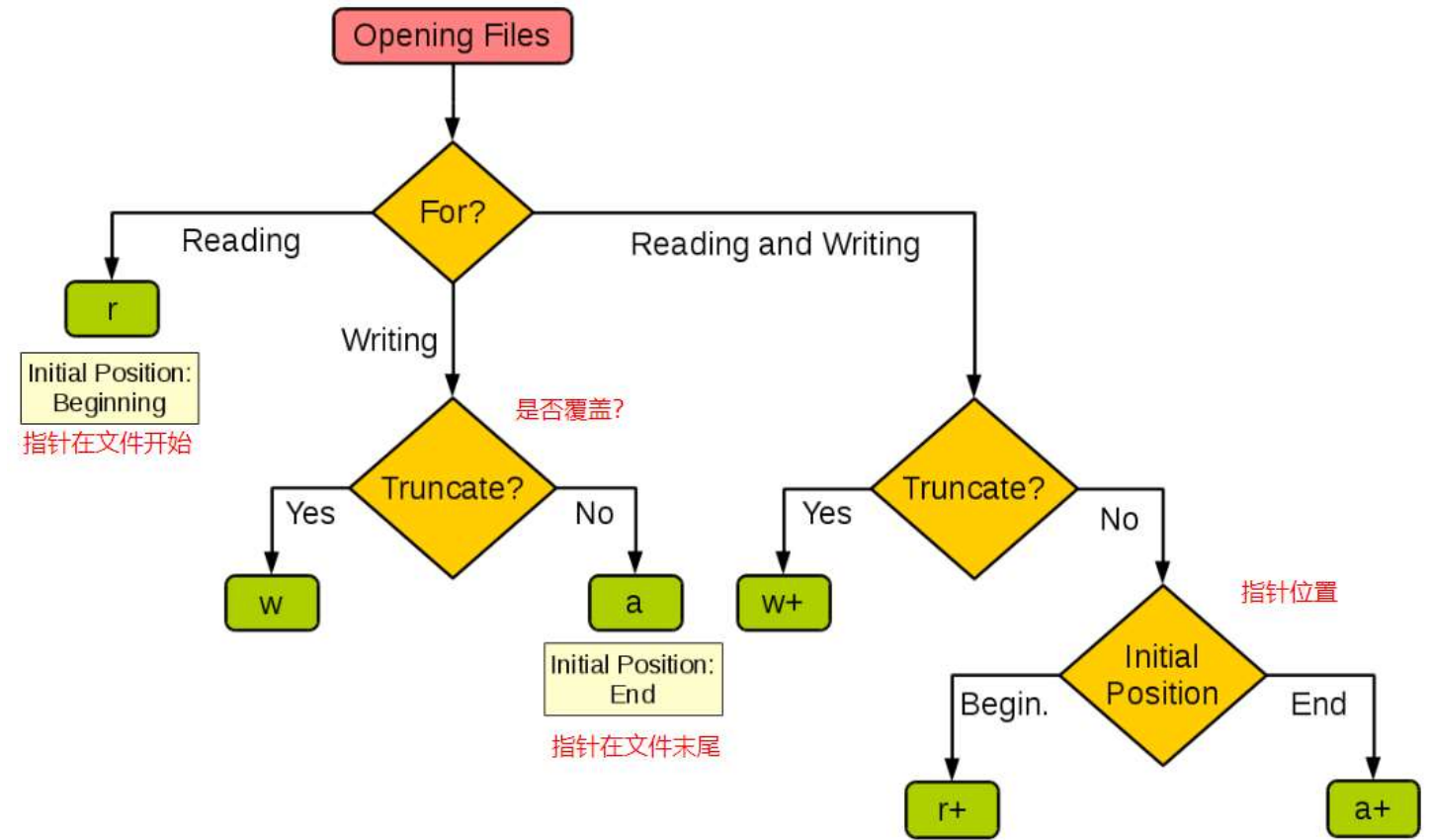
```
with open('处理所需文件/pi_million_digits.txt') as file_object:
    lines = file_object.readlines()
    content = ""
    my_birthday = input("Your Birthday:")
    for line in lines:
        content += line
    if my_birthday in content:
        print("yes")
    else:
        print("no")
```

2、写入文件

2020/1/13

文件和异常

打开文件时，可指定读取模式 'r'、写入模式 'w'、附加模式 'a' 或让你能够读取和写入文件的模式 'r+'。如果你省略了模式实参，Python 将以默认的只读模式打开文件。下图很好的展示了不同情况下如何选择打开和写入方式：



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

查看更多

```
1 # I love programming.
2 # I love creating new games.
3 # I also love finding meaning in large datasets.
4 # I love creating apps that can run in a browser.
5 # 该方式写入文件，如果写入前存在文件，文件会清空，再写入，具体情况看上表
6 with open('处理所需文件/programming.txt', 'w') as write_file:
7     write_file.write('I love programming.')
```

注意：

Python 只能将字符串写入文本文件。要将数值数据存储到文本文件中，必须先使用函数 `str()` 将其转换为字符串格式。

3、异常

Python 实践中：

异常是使用 try-except 代码块处理的。try-except 代码块让 Python 执行指定的操作，同时告诉 Python 发生异常时怎么办。使用了 try-except 代码块时，即便出现异常，程序也将继续运行：显示你编写的友好的错误消息，而不是令用户迷惑的 traceback。

3.1 异常使用

当我们编写代码经常会出错，但是出错就使程序崩溃有时候不太好，但让用户看到 traceback 也不是好主意。不懂技术的用户会被它们搞糊涂，而且如果用户怀有恶意，他会通过 traceback 获悉你不希望他知道的信息。例如，他将知道你的程序文件的名称，还将看到部分不能正确运行的代码。有时候，训练有素的攻击者可根据这些信息判断出可对你的代码发起什么样的攻击。

```
a = 0
b = 5
# 报异常，程序也结束了 ZeroDivisionError: division by zero
# c = b/a
c = 0
try:
    c = b / a
# a == 0 抛出一个友好的异常
except ZeroDivisionError:
    print("You can't divide by 0!")
else:
    print("continue!")
print(c)
```

try-except-else 代码块的工作原理大致如下：Python 尝试执行 try 代码块中的代码；只有可能引发异常的代码才需要放在 try 语句中。有时候，有一些仅在 try 代码块成功执行时才需要运行的代码；这些代码应放在 else 代码块中。except 代码块告诉 Python，如果它尝试运行 try 代码块中的代码时引发了指定的异常，该怎么办。

一个统计文章单词的例子

```
try:
    with open('处理所需文件/45.txt') as ani:
        text = ani.read()
except FileNotFoundError:
    print('文件没有找到! ')
else:
    words = text.split();
    length = len(words)
    print("文章长度为: "+str(length))
```

3.2 使用多个文件

将查找单词的方法放入函数，然后调用统计多个文件的字数

```
1 | from CountWords import count_words
2 |
```

```
3 file_paths = ['处理所需文件/45.txt', '处理所需文件/alice.txt', '处理所需文件/ali', '处理所需文件/siddhartha.txt']
4 for file_path in file_paths:
5     print(count_words(file_path))
```

在这个示例中，使用 try-except 代码块提供了两个重要的优点：

- 避免让用户看到 traceback
- 让程序能够继续分析能够找到的其他文件

3.3 失败时一声不吭

当程序出现异常的时候，我们如果希望程序不希望做出任何的反应的时候，可以使用 `pass` 关键字

```
1 try:
2     c = 5 / 0
3 except ZeroDivisionError:
4     pass
5 else:
6     print("continue!")
7 print('emmm')
```

4、存储数据

模块 json 让你能够将简单的 Python 数据结构转储到文件中，并在程序再次运行时加载该文件中的数据。你还可以使用 json 在 Python 程序之间分享数据。

4.1 使用 json.dump()和 json.load()

函数 `json.dump()` 接受两个实参：

- 要存储的数据
- 可用于存储数据的文件对象

下面演示了如何使用 `json.dump()` 来存储数字列表：

```
1 import json
2
3 names = ['小王', '小李', '老王', '老李']
4 file_name = '处理所需文件/names.txt'
5
6 with open(file_name, 'w') as json_write:
7     json.dump(names, json_write)
```

函数 `json.load()` 接受一个参数：

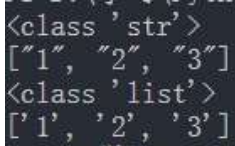
- 要读取的文件对象

```
1 import json
2
3 file_name = '处理所需文件/names.txt'
4 with open(file_name) as json_read:
5     names2 = json.load(json_read)
6     for name in names2:
7         print(name)
8
```

4.2 json.dumps()和json.loads()

dumps()将Python对象转为json字符串，loads()反之。

```
1 import json
2
3 numbers = ['1', '2', '3']
4
5 numbers = json.dumps(numbers)
6 print(type(numbers))
7 print(numbers)
8 numbers = json.loads(numbers)
9 print(type(numbers))
10 print(numbers)
```



```
<class 'str'>
["1", "2", "3"]
<class 'list'>
['1', '2', '3']
```