第九章：输入输出与文件

Table of Contents

[1. 序言 2](#_Toc51528590)

[1.1 章节简介 2](#_Toc51528591)

[2. 文件读写 2](#_Toc51528592)

[2.1 open() 2](#_Toc51528593)

[2.2 write() 4](#_Toc51528594)

[2.3 read() 4](#_Toc51528595)

[3. 操作文件和目录 5](#_Toc51528596)

[4. 序列化 6](#_Toc51528597)

[4.1 pickle 7](#_Toc51528598)

[4.2 JSON 8](#_Toc51528599)

# 序言

## 章节简介

IO在计算机中是指input/output，也就是输入和输出。由于程序和运行时数据是在内存中驻留，由CPU这个超快的核心来执行，涉及到数据交换的地方，通常是磁盘，网络等，这里就需要IO接口。通常，程序完成IO操作会有input和output两个数据流。在IO编程中stream是一个很重要的概念，顾名思义stream就是一根里面由水流流过的水管，数据就是里面的水，但是只可以单向流动。Input Stream就是数据从外面（磁盘、网络）流进内存，Output Stream就是数据从内存流到外面去。

另外一个值得注意的问题就是CPU和内存的速度远远高于外设的速度，因此在IO编程中存在速度严重不匹配的问题。为了解决这个问题，有两种方法：同步IO和异步IO。这里讲的主要是同步IO，异步复杂度较高。本章主要介绍一些Python中IO的简单操作，具体内容为：

a. 文件读写

b. 操作文件和目录

c. 序列化

# 文件读写

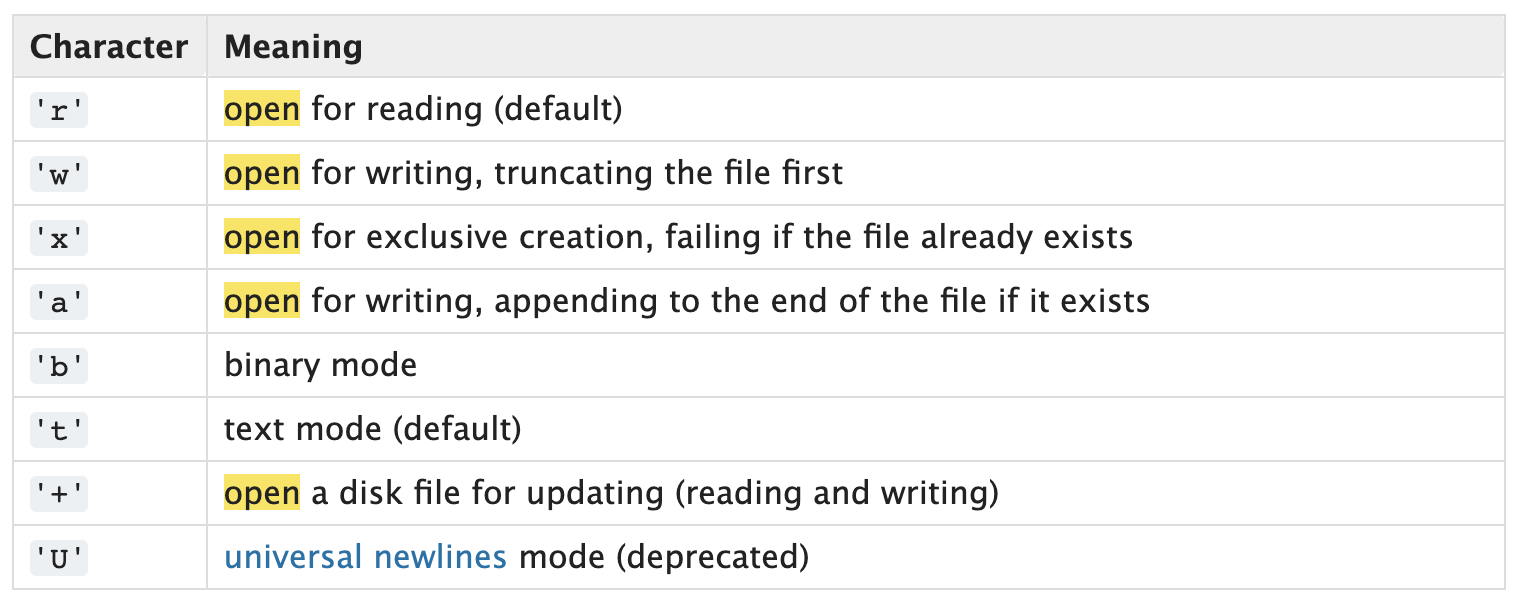
首先来看看怎么读写实际的数据文件。Python 提供了必要的函数和方法进行默认情况下的文件基本操作。你可以用 file 对象做大部分的文件操作。

## open()

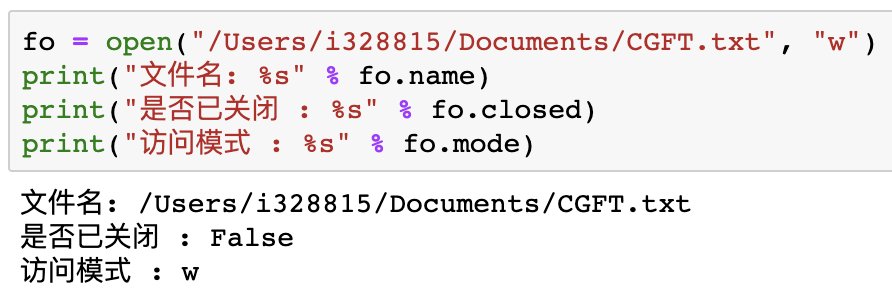
我们先看open函数，你必须先用Python内置的open()函数打开一个文件，创建一个file对象，相关的方法才可以调用它进行读写。使用open()语法如下：

*file object = open(file\_name [, access\_mode][, buffering])*

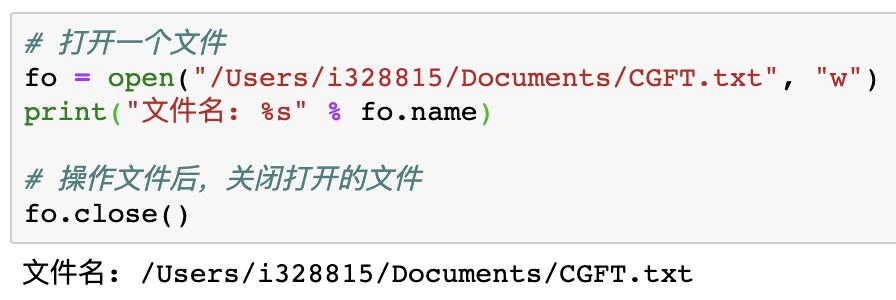
各个参数的细节如下：file\_name：file\_name变量是一个包含了你要访问的文件名称的字符串值。access\_mode：access\_mode决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。buffering:如果buffering的值被设为0，就不会有寄存。如果buffering的值取1，访问文件时会寄存行。如果将buffering的值设为大于1的整数，表明了这就是的寄存区的缓冲大小。如果取负值，寄存区的缓冲大小则为系统默认。关于访问模式的参数介绍如下图示，更多的详细可以参见Python的官方网站<https://docs.python.org/3.6/library/functions.html?highlight=open#open>：



当我们用open方法打开一个文件时，我们就有了一个file对象，你可以得到有关该文件的各种信息。如下示例：



操控文件时还有一个对应的close()方法，File 对象的 close()方法刷新缓冲区里任何还没写入的信息，并关闭该文件，这之后便不能再进行写入。文件使用完毕后必须关闭，因为文件对象会占用操作系统的资源，并且操作系统同一时间能打开的文件数量也是有限的。用 close()方法关闭文件是一个很好的习惯。



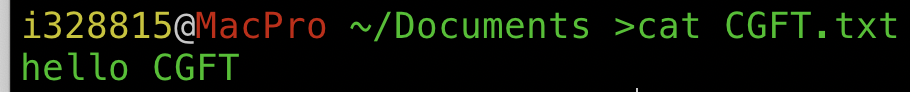
## write()

对于读写文件，file对象提供了一系列方法，能让我们的文件访问更轻松。来看看如何使用read()和write()方法来读取和写入文件。

首先是write()方法，write()方法可将任何字符串写入一个打开的文件。需要重点注意的是，Python字符串可以是二进制数据，而不是仅仅是文字。



上述方法首先打开我的电脑上的CGFT.txt文件，并将”hello CGFT”写入该文件，并最终关闭文件。如果你打开这个文件，将看到以下内容(下面使用Linux命令打开文件，同学们可以通过文本编辑器在自己的电脑上查看文本文件的内容，没有区别):



## read()

read()方法从一个打开的文件中读取一个字符串。需要重点注意的是，Python字符串可以是二进制数据，而不是仅仅是文字。语法：*fileObject.read([count])*

在这里，被传递的参数是要从已打开文件中读取的字节计数。该方法从文件的开头开始读入，如果没有传入count，它会尝试尽可能多地读取更多的内容，很可能是直到文件的末尾。下面的示例读取我们刚才编辑过得文件CGFT.txt,注意open()方法的第二个参数是’r’，表示的是获取一个文件，用于读取(read)：



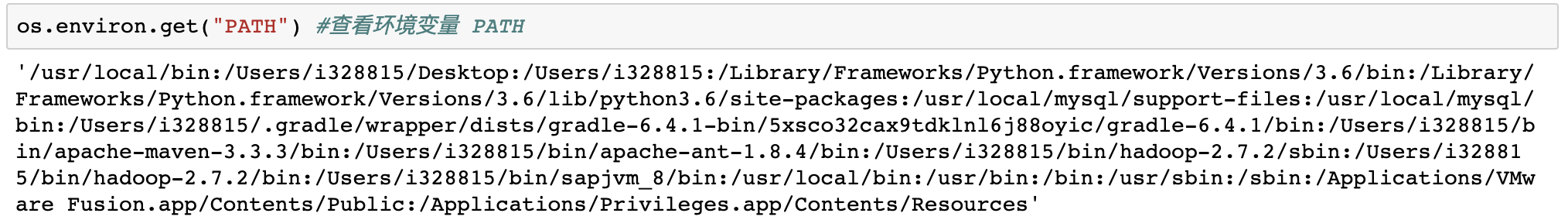
# 操作文件和目录

在python中对文件及目录的操作一般涉及多os模块(关于模块会在11章详细介绍)，os.path模块。我们来看看操作os的基本功能。

os模块首先可以查看操作系统的信息：



有开发经验或者使用过环境变量的同学可能知道，在操作系统中定义的环境变量，全部保存在os.environ这个变量中，也可以在Python中直接查看：



操作文件和目录的函数一部分放在os模块中，一部分放在os.path模块中。下面的示例展示了如何通过Python查看、创建目录：



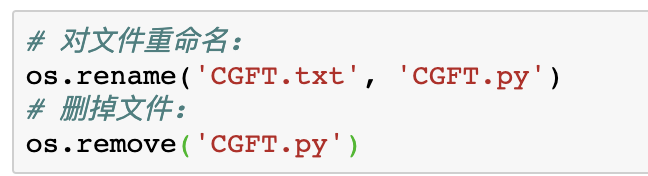
注意上面的示例中，把两个路径合成一个时，不要直接手动输入路径的字符串，而是通过os.path.join()函数。这是因为在Linux/Unix/Mac操作系统下，os.path.join()返回的字符串像下面这种：/Users/i328815/Documents。而在Windows操作系统里，表示路径的字符串像下面这种：“C:\Users\Documents“。Python会帮助我们正确处理不同操作系统的路径分隔符。下面的示例是删除文件夹：



我们还可以通过os.path.split()函数，这样可以把一个路径进行拆分，得到路径和最后级别的目录或文件名：



我们继续操作文件，可以用下面的方法给文件改名或删除文件。



# 序列化

序列化 (pickling)是指将对象、数据结构的状态信息转换为可以存储或传输的形式的过程。在序列化期间，对象将其当前状态写入到临时或持久性存储区。以后，可以通过从存储区中读取或反序列化对象的状态，重新创建该对象。我们编写的程序，会涉及到各种各样的对象、数据结构，它们通常是以变量的形式在内存中存在着。当程序运行结束后，这些变量也就会被清理。但我们有时希望能够在下一次编写程序时恢复上一次的某个对象（如机器学习中的到结果，需要程序运行较长时间，多次运行时间成本太大），这就需要我们将变量进行持久化的存储。一种方式是利用文件读写的方式将变量转化为某种形式的字符串写入文件内，但需要自己控制存储格式显得十分笨拙。更好的方式是通过序列化的方式将变量持久化至本地。本小节我们主要学习Python中的序列化问题，Python中的序列化主要是pickle和JSON，我们来分别介绍。

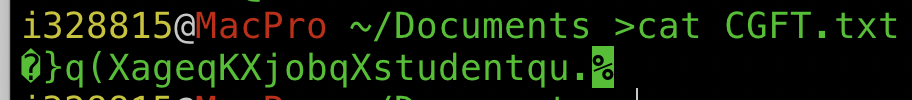
## pickle

我们先来看看如何将内存中的变量存储在文件中。使得对象持久化。使用的是dump和load函数。由于pickle写入的是二进制数据，所以打开方式需要以wb和rb的模式。

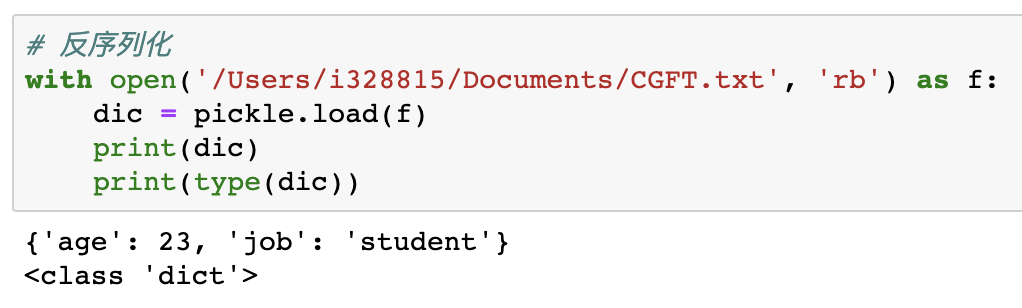
下面的示例把一个字典变量以字节的方式存储在文件中：



当我们查看文件，会看到里面的内容存储了乱码，这些是存储的字典的信息：



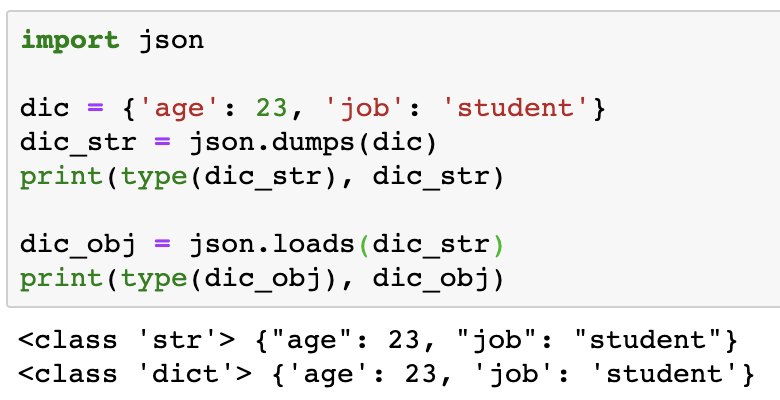
我们再来反序列化，即将磁盘上存储的字节内容读取到内存中并转换回对象：



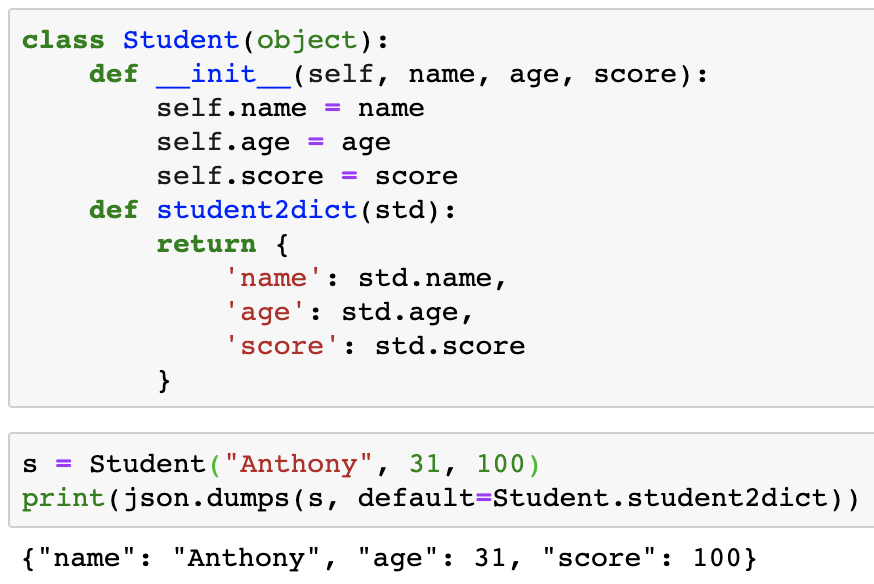
## 4.2 JSON

pickle可以很方便地序列化所有对象。不过json作为更为标准的格式，表示出来就是一个字符串，可以被所有语言读取，也可以方便地存储到磁盘或者通过网络传输。JSON不仅是标准格式，并且比XML更快，而且可以直接在Web页面中读取，非常方便。具有更好的可读性（pickle是二进制数据）和跨平台性。是个不错的选择。

在下面的代码中，dumps()方法返回一个str，内容就是标准的JSON。另一方面可以使用用loads()把JSON格式的字符串反序列化为Python对象：



我们在实际编程中，往往需要把类的对象序列化进行传输或存储，我们同样可以通过json完成。为了让类对象可以进行json序列化，我们需要为Student专门写一个转换函数，再把函数传进去给json模块的dumps()方法即可，详见下面示例。关于json.dump函数的详细使用，可以参见Python官方文档<https://docs.python.org/3/library/json.html#json.dump> ：



反序列化时，如果我们要把JSON格式的字符串反序列化为一个Student对象实例，loads()方法首先转换出一个dict对象，然后，我们传入的object\_hook函数负责把dict转换为Student实例：

