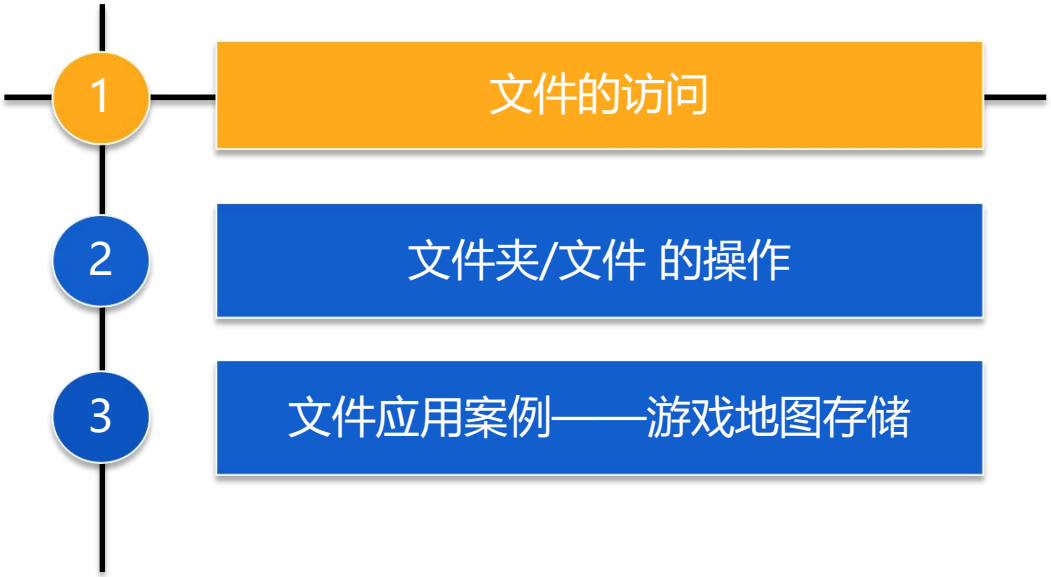
Python程序设计
—从基础到开发

目录

第5章 Python文件的使用



5.1 文件的访问

文件

- 程序运行时，数据保存在内存的变量中，程序结束就会消失；
- 想要使用相同的数据，需要把数据存储在非易失的存储介质中；
- 在非易失介质上存储的数据以文件形式存储；
- 文件有路径名；
- 运行时的程序，通过读/写文件，来实时保存数据或获取数据。

- 简单地说，文件是由字节组成的信息，在逻辑上具有完整意义，通常在磁盘上永久保存。

5.1 文件的访问

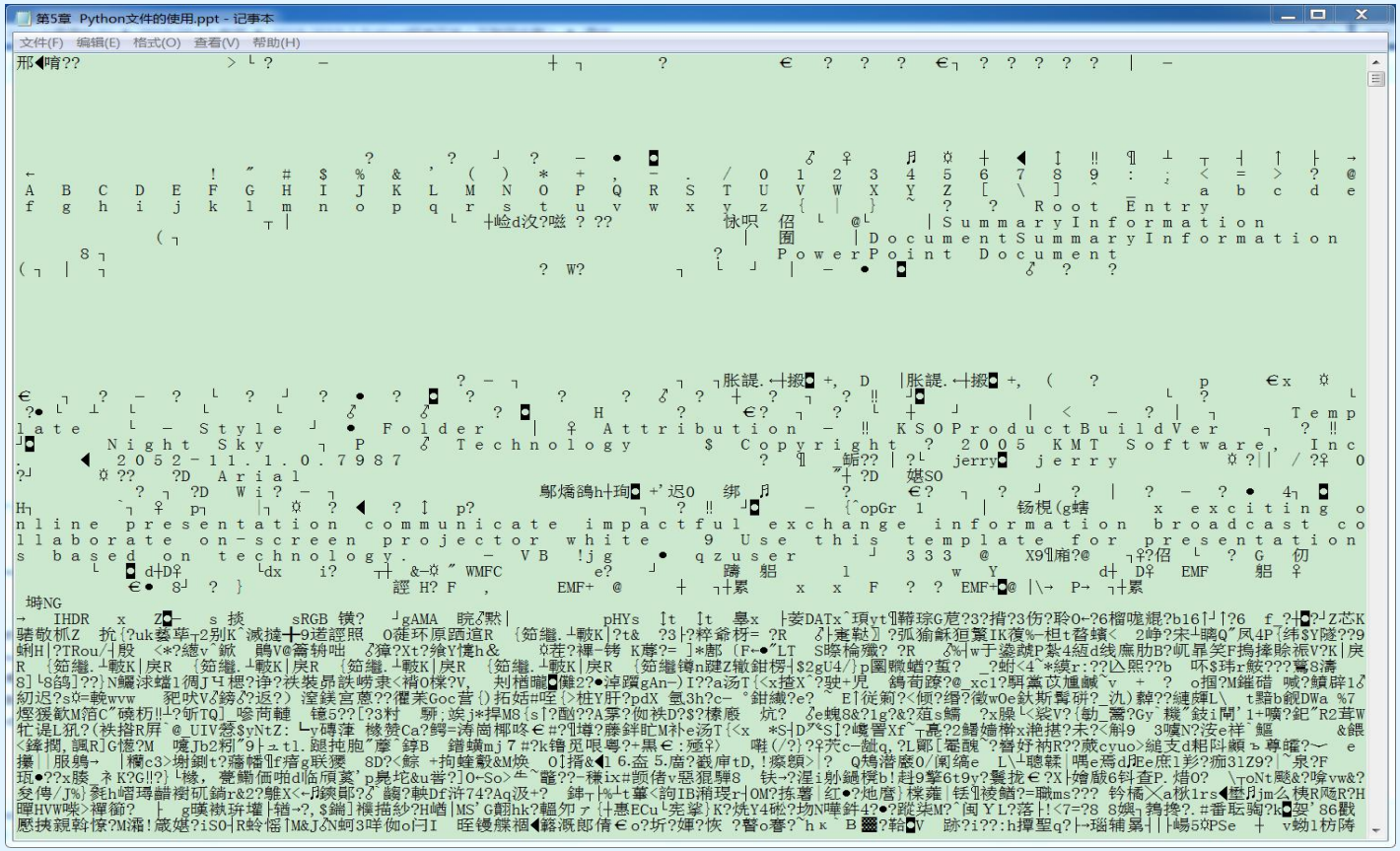
文件

- Windows系统的数据文件按照编码方式分为两大类：
  - 文本文件
  - 二进制文件。
- 文本文件可以处理各种语言所需的字符，只包含基本文本字符，不包括诸如字体、字号、颜色等信息。它可以在文本编辑器和浏览器中显示。
  - 即在任何情况下，文本文件都是可读的。
- 使用其它编码方式的文件即二进制文件，如Word文档、PDF、图像和可执行程序等。

5.1 文件的访问

文件

- 二进制文件一般都需要自己的处理程序才能打开或操作。



5.1 文件的访问

在Python中对文件的操作通常按照以下三个步骤进行：

（1）使用open()函数打开（或建立）文件，返回一个file对象。

（2）使用file对象的读/写方法对文件进行读/写的操作。

- 将数据从外存传输到内存的过程称为读操作；
- 将数据从内存传输到外存的过程称为写操作。

（3）使用file对象的close()方法关闭文件。

5.1 文件的访问

5.1.1 打开（建立）文件 —— open()函数

- open() 函数需要一个字符串路径，表明希望打开文件，并返回一个文件对象。文件对象有时又称为文件描述符或文件流。
- 语法：fileobj=open(filename[, mode[, buffering]])

```
>>> helloFile=open("d:\\python\\hello.txt")
```

- 参数 filename 必须有，可以是绝对路径，也可以是相对路径。
- 参数 mode、buffering 可选。

5.1 文件的访问

5.1.1 打开（建立）文件 —— open()函数

- open()函数语法：fileobj=open(filename[, **mode**[, buffering]])
- open函数中**mode**参数常用值

值	描述
'r'	读模式，如果文件不存在，则发生异常
'w'	写模式，如果文件不存在，则创建文件再打开， 如果文件存在，则清空文件内容再打开
'a'	追加模式，如果文件不存在，则创建文件再打开， 如果文件存在，打开后新内容追加到原内容之后
'b'	二进制模式，可添加到其他模式中使用，如 'rb'
'+'	读/写模式，可添加到其他模式中使用，如 'w+'

5.1 文件的访问

5.1.1 打开（建立）文件 —— open()函数

- open()函数语法：fileobj=open(filename[, mode[, **buffering**]])
- open函数中**buffering**参数控制文件操作是否使用缓冲。

值	描述
'0' False	写操作没有缓冲，直接针对硬盘
'1' True	写有缓冲，使用flush或者close时，才会将数据写入硬盘
> 1	表示缓冲区大小，以字节为单位
< 0	表示使用默认大小缓冲区大小

5.1 文件的访问

5.1.2 读取文本文件 —— read()，readline()，readlines()

1．read() 方法

```
helloFile=open("d:\\python\\hello.txt")
fileContent=helloFile.read()
helloFile.close()
print(fileContent)
```

- 无参数调用read()方法，会将整个文件的内容读取为一个字符串。
  - 一次性读取文件全部内容
  - 性能随文件大小变化，需要同文件大小一样的内存空间。

5.1 文件的访问

5.1.2 读取文本文件 —— read()，readline()，readlines()

1．read() 方法

- 如果文件太大，也可以设定一次最大读入字符数。

```
helloFile=open("d:\\python\\hello.txt")
fileContent = ' '
while True:
    fragment = helloFile.read(3)
    if not fragment:          #如果返回空字符串，说明到文件尾
        break
    fileContent += fragment
helloFile.close()
print(fileContent)
```

5.1 文件的访问

5.1.2 读取文本文件 —— read()，readline()，readlines()

2．readline() 方法

- readline()方法从文件中获取文件中的每一行

```
helloFile=open("d:\\python\\hello.txt")
fileContent=""
while True:
   line=helloFile.readline()
   if line=="":     # 或者 if not line
      break
   fileContent+=line
helloFile.close()
print(fileContent)
```

5.1.2 读取文本文件 —— read()，readline()，readlines()

3．readlines() 方法

- readlines()方法返回一个字符串列表，其中每一个元素是文件中的每一行

5.1 文件的访问

5.1.3 写文本文件 —— write ()

- 写文件与读文件相似，都需要先创建文件对象连接。不同的是，打开文件时是以“w”或“a”模式打开。如果文件不存在，则创建该文件。
- 与读文件时不能添加或修改数据类似，**以写方式打开文件，也不允许读取数据**。

```
>>> helloFile=open("d:\\python\\hello.txt","w")
>>> fileContent = helloFile.read()
UnsupportedOperation: not readable
>>> helloFile.close()
```

- “w”模式打开已有文件时，会覆盖文件原有内容，从头开始，就像我们用一个新值覆写一个变量的值。

```
>>> helloFile=open("d:\\python\\hello.txt")
>>> fileContent = helloFile.read()
>>> len(fileContent)
0
>>> helloFile.close()
```

5.1 文件的访问

5.1.3 写文本文件 —— write ()

- write方法将字符串参数写入文件。

```
>>> helloFile=open("d:\\python\\hello.txt","w")
>>> helloFile.write("First line. \nSecond line. \n")
>>> helloFile.close()
>>> helloFile=open("d:\\python\\hello.txt","a")
>>> helloFile.write("third line.")
>>> helloFile.close()
>>> helloFile=open("d:\\python\\hello.txt")
>>> fileContent = helloFile.read()
>>> helloFile.close()
>>> print(fileContent)
```

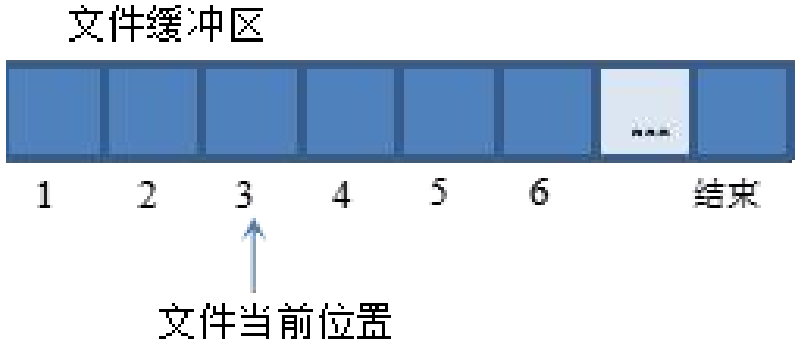
- —— write ()

输出结果：
First line.
Second line.
third line.

5.1 文件的访问

5.1.4 文件内移动

- 无论读或写文件，Python都会跟踪文件中的读写位置。在默认情况下，文件的读/写都从文件的开始位置进行。
- 当使用open()函数打开文件时，会在内存中从创建缓冲区，将磁盘上的文件内容复制到缓冲区。



- 然后文件对象将缓冲区视为一个大的列表，每个字节都是这个列表中的一个元素，每个元素都有自己的索引。同时，文件对象对文件当前位置进行维护。Python使用一些函数跟踪文件当前位置。
  - tell() 函数可以返回文件当前位置距首字节的偏移量。

5.1 文件的访问

5.1.4 文件内移动 —— tell()

```
>>> exampleFile=open("d:\\python\\example.txt","w")
>>> exampleFile.write("0123456789")
>>> exampleFile.close()
>>> exampleFile=open("d:\\python\\example.txt")
>>> exampleFile.read(2)
'01'
>>> exampleFile.read(2)
'23'
>>> exampleFile.tell()     #表示文件当前位置和开始位置之间的偏移
4
>>> exampleFile.close()
```

5.1 文件的访问

5.1.4 文件内移动 ——seek()

- seek() 函数将文件指针由当前位置移动到指定位置。
- 即设置新的文件当前位置，允许在文件中跳转，实现对文件的随机访问。
- 语法格式：seek(offset[, whence])
  - 第一个参数是字节数（偏移量）
  - 第二个参数是offset的引用点
    - 0：文件开始处
    - 1：文件当前位置
    - 2：文件末尾

5.1 文件的访问

5.1.4 文件内移动

- 用seek()函数在指定位置写文件

```
>>> exampleFile=open("d:\\python\\example.txt","w")
>>> exampleFile.write("0123456789")
>>> exampleFile.seek(3)              #缺省引用点在文件开始处
>>> exampleFile.write("ZUT")
>>> exampleFile.close()
>>> exampleFile=open("d:\\python\\example.txt")
>>> s = exampleFile.read()
>>> print(s)
>>> exampleFile.close()
'012ZUT6789'
```

5.1 文件的访问

5.1.5 文件的关闭—— close()

- 关闭文件是取消程序和文件之间连接的过程，内存缓冲区的所有内容将写入磁盘，因此必须在使用文件后关闭文件确保信息不会丢失。
- **应该牢记使用close方法关闭文件。**

```
helloFile=open("d:\\python\\hello.txt","w")
try :
    helloFile.write("Hello,Sunny Day!")
finally:
    helloFile.close()
```

```
#也可以使用with语句自动关闭文件：
with open("d:\\python\\hello.txt") as helloFile:
    s=helloFile.read()
print(s)
```

5.1 文件的访问

5.1.6 二进制文件的读写

- 有的时候需要用python处理二进制数据。比如，存取文件，socket操作；
- Python没有二进制类型，用string字符串类型来存储二进制类型数据
  - 因为string是以字节为单位的；
  - 可以使用python的struct模块来完成。
    - python中的struct模块主要是用来处理C结构数据的
    - 读入时先转换为Python的字符串类型
    - 然后再转换为Python的结构化类型，比如元组(tuple)等。
  - struct模块中最重要的三个函数是：pack(), unpack(), calcsize()

5.1 文件的访问

5.1.6 二进制文件的写

- 首先使用struct.pack将数据转换成字节串（以字节为单位的字符串）
- 然后将字节串写入文件，例如：

```
import struct
a='hello'
b='world!'
c=2
d=45.123
bytes=struct.pack('5s6sif', a.encode('utf-8'), b.encode('utf-8'), c, d)
print(bytes)
```

假设要写入文件的数据是由多个数据构成的

为了4字节对齐，填充的字节

结果输出：b'helloworld!\x00\x02\x00\x00\x00\xf4}4B'

```
binfile = open("d:\\python\\hellobin.txt","wb")
binfile.write(bytes)        #把字符串写入到二进制文件中
binfile.close()
```

5.1 文件的访问

5.1.6 二进制文件的读写

- 一般输入（读入）的渠道来源于文件或者网络的二进制流
  - pack() 方法可以把数据转换成字节串；

    格式：struct.pack(fmt,v1,v2,.....)
  - unpack() 方法可以把相应数据的字节串还原成数据；

    格式：struct.unpack(fmt, string)
  - calcsize(fmt) 计算给定的格式(fmt)占用多少字节的内存。

5.1 文件的访问

5.1.6 二进制文件的读

- 字节串（以字节为单位的字符串）还原成数据
- 首先以二进制读"rb"方式打开文件，然后读取文件到字节串中
- 然后使用struct.unpack将字节串还原成数据，例如：

```
import struct
binfile=open("d:\\python\\hellobin.txt","rb")
bytes=binfile.read()
(a,b,c,d)=struct.unpack('5s6sif',bytes)  #通过struct.unpack()解码成python变量
#或者
t=struct.unpack('5s6sif',bytes)          #通过struct.unpack()解码成元组
print(t)
```

5.1 文件的访问

格式字符

格式字符	C语言中的类型	Python的类型	字节数
c	char	string of length 1	1
b	signed char	integer	1
B	unsigned char	integer	1
?	_Bool	bool	1
h	short	integer	2
H	unsigned short	integer	2
i	int	integer	4
I	unsigned int	integer	4

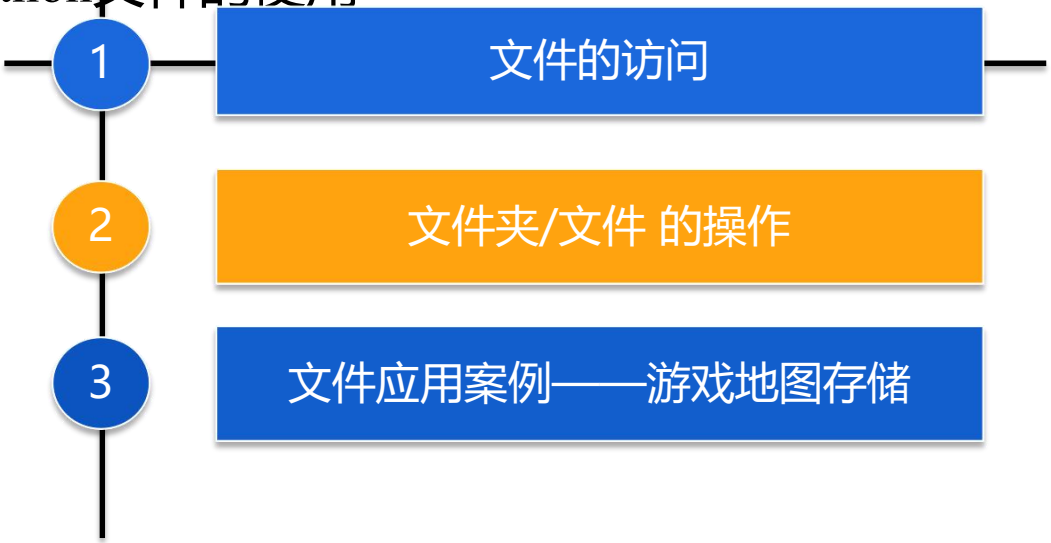
5.1 文件的访问

格式字符

格式字符	C语言中的类型	Python的类型	字节数
l	long	integer	4
L	unsigned long	integer	4
q	long long	integer	8
Q	unsigned long long	integer	8
f	float	float	4
d	double	float	8
s	char[]	string	1
p	char[]	string	1
P	void *	integer	与OS有关

目录

- 第5章 Python文件的使用



5.2 文件夹/文件的操作

- 文件有两个关键属性：路径和文件名
  - 路径指明了文件在磁盘上的位置。
  - 文件名后面部分有扩展名，指明了文件的类型。

- Python有两个模块与文件及文件夹操作紧密相关
  - os 模块
  - shutil 模块（**sh**ell **util**ity）

5.2 文件夹/文件的操作

5.2.1 当前工作目录

- 每个运行在计算机上的程序，都有一个“当前工作目录”；
- 所有没有从根文件夹开始的文件名或路径，都假定工作在当前目录下。
- 使用 **os.getcwd()** 函数可获得当前工作目录：

```
>>> import os
>>> os.getcwd()
```

修改当前工作目录

- 使用 **os.chdir()** 函数可以更改当前工作目录：

```
>>> os.chdir("e:\\python1")
>>> os.listdir(".")          # . 代表当前工作目录
```

5.2 文件夹/文件的操作

5.2.2 创建新目录

- 可以用 os.makedirs() 函数创建新目录。在交互式环境中输入以下代码：

```
>>> import os
>>> os.makedirs("e:\\python1\\ch5files")          #创建多级目录
>>> os.mkdir("e:\\python1\\ch5files2")            #创建单个目录
```

5.2 文件夹/文件的操作

5.2.3 列出目录内容

- **os.listdir()** 函数可以返回“**路径中**”文件名及文件夹名的字符串列表：

```
>>> os.listdir("e:\\python1")
```

输出： **['ch5files', 'ch5files2']**

5.2.4 获取文件大小

- os.path 模块中的 os.path**.getsize()** 函数可以获取文件大小。

5.2 文件夹/文件的操作

5.2.5 删除文件夹/文件

os模块和shutil模块都有函数可以删除文件或文件夹。

- **os.rmdir(path)**：可以删除目录

```
>>> os.rmdir("e:\\python1\\ch5files2")
```

  - **注意：**rmdir() 函数只能删除空文件夹，如果文件夹非空，会报错。

- **shutil.rmtree(path)**：可以删除整个文件夹，包含所有文件及子文件夹

- **os.remove(path)/os.unlink(path)**：删除参数path指定的文件。

```
>>> os.remove("e:\\python-backup\\data-backup.txt")          #删除文件
>>> os.path.exists("e:\\python-backup\\data-backup.txt")     #输出False
```

5.2 文件夹/文件的操作

5.2.6 获取路径和文件名

在编程中经常会使用到文件的属性，例如文件路径、文件名等，这些可以使用 **os.path模块** 中的函数获取。

- os.path.**dirname(path)**：返回path参数中的**路径名称**字符串。
- os.path.**basename(path)**：返回path参数中的**文件名**。
- os.path.**split(path)**：返回参数的路径名称和文件名组成的字符串元组。

5.2 文件夹/文件的操作

5.2.7 检查路径有效性

- os.path.**exists(path)**：判断参数path的文件或文件夹是否存在。
  - 存在返回true，否则返回false。
- os.path.**isfile(path)**：判断参数path是否存在且是否是一个文件，
  - 存在且是文件则返回true，否则返回false。
- os.path.**isdir(path)**：判断参数path存在且是一个文件夹
  - 存在且是文件夹则返回true，否则返回false。

5.2 文件夹/文件的操作

5.2.8 文件和文件夹的改名、移动、复制

- **os.rename(source, destination)**：可以重命名文件

```
>>> os.rename("d:\\python\\hello.txt","d:\\python\\helloworld.txt")
```

- **shutil.move(source, destination)**：移动文件夹或文件
  - 参数destination 既可以是一个包含新文件名的路径，也可以仅包含文件夹。
  - shutil.move() 函数与shutil.copy() 函数用法相似

- **shutil.copy(source, destination)**：复制文件
- **shutil.copytree(source, destination)**：复制整个文件夹，包括文件及子文件夹

5.2 文件夹/文件的操作

5.2.9 遍历目录树

- 想要处理文件夹中包括子文件夹内的所有文件即遍历目录树，可以使用os.walk()函数。os.walk()函数返回该路径下所有文件及子目录信息元组。

```
import os
list_dirs = os.walk("K:\\2020-09-01 教学\\2020-2021-1 Python程序开发")  #返回一个元组
print(list(list_dirs))
for folderName,subFolders,fileNames in list_dirs:
    print("当前目录： "+ folderName)
    for subFolder in subFolders:
        print(folderName + "的子目录是——"+subFolder)
    for fileName in fileNames:
        print(folderName+“目录下的文件是——"+ fileName)
```

目录

第5章 Python文件的使用

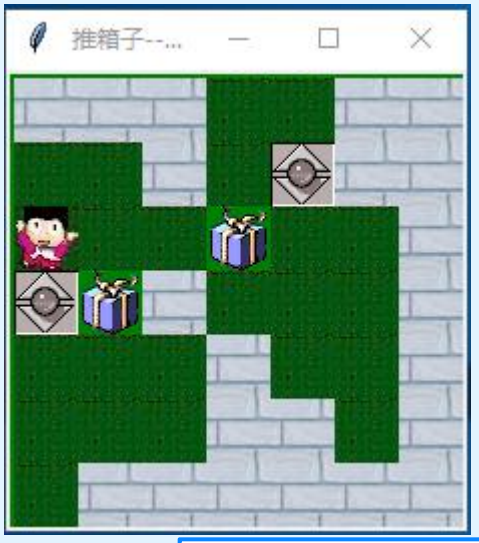
5.3 文件应用案例——游戏地图存储

- 在游戏开发中往往需要存储不同关卡的游戏地图信息，例如推箱子，连连看等游戏。
- 本小节以推箱子游戏地图的存储来说明游戏地图信息如何存储到文件中，并读取出来。如图可以看成一个7*7的表格
- 如果按行存储到文件中，就可以把某关的地图存入到文件中了。
- 为了表示方便，每个格子的状态用如下常量表示：
  - Wall(0)代表墙、Worker(1)代表人
  - Box(2)代表箱子、Passageway(3)代表路
  - Destination(4)代表目的地
  - WorkerInDest(5)代表人在目的地
  - RedBox(6)代表放到目的地的箱子。

5.3 文件应用案例——游戏地图存储

- 在游戏开发中往往需要存储不同关卡的游戏地图信息，例如推箱子，连连看等游戏。
- 本小节以推箱子游戏地图的存储来说明游戏地图信息如何存储到文件中，并读取出来。如图可以看成一个7*7的表格
- 如果按行存储到文件中，就可以把某关的地图存入到文件中了。
- 为了表示方便，每个格子的状态用如下常量表示：
  - Wall(0)代表墙、Worker(1)代表人
  - Box(2)代表箱子、Passageway(3)代表路
  - Destination(4)代表目的地
  - WorkerInDest(5)代表人在目的地
  - RedBox(6)代表放到目的地的箱子。

5.3 文件应用案例——游戏地图存储



墙	墙	墙	3	3	墙	墙
3	3	墙	3		墙	墙
	3	3		3	3	墙
		墙	3	3	3	墙
3	3	3	墙	3	3	墙
3	3	3	墙	墙	3	墙
3	墙	墙	墙	墙	墙	墙

0	0	0	3	3	0	0
3	3	0	3	4	0	0
1	3	3	2	3	3	0
4	2	0	3	3	3	0
3	3	3	0	3	3	0
3	3	3	0	0	3	0
3	0	0	0	0	0	0

```
import os
mapFile=open("map1.txt","w")
mapFile.write("0,0,0,3,3,0,0\n")
mapFile.write("3,3,0,3,4,0,0\n")
mapFile.write("1,3,3,2,3,3,0\n")
mapFile.write("4,2,0,3,3,3,0\n")
mapFile.write("3,3,3,0,3,3,0\n")
mapFile.write("3,3,3,0,0,3,0\n")
mapFile.write("3,0,0,0,0,0,0\n")
mapFile.close()
```

```
mapFile=open("map1.txt","r")
myArray1 = []
while True:
  line=mapFile.readline()
  if line=="":    # 或者 if not line
    break
  line=line.replace("\n","")
  myArray1.append(line.split(","))
mapFile.close()
print(myArray1)
```