复习

进制相关

0b 二进制， 0o 八进制 0x 十六进制， box 盒子

bin() 参数是整形，返回整形对应的二进制字符串

oct() 参数是整形，返回对应的八进制字符串

hex() 参数是整形，返回对应的十六进制字符串

另一种

```
int(x, base=10)
x -- 数字。
base -- 进制数，默认十进制。

n = int('10',base=16)
print(n)
n = int('10',base=8)
print(n)
```

用法：

首先： 0x56是整形吗?

```
print(type(0x56))
<class 'int'>
是整形，只是以16进制的形式表示，所以。。。。
bin() 参数是整形，返回整形对应的二进制字符串
print(type(bin(0x0a))) <class 'str'>  不会报错
```

进制转换应该没问题吧

文件相关

CSV格式 ， \n

open, close

不关不行!

open() 函数常用形式是接收两个参数：文件名(file)和模式(mode)。返回一个文件对象

mode:

r 只读，文件指针指向开头

r+ 打开一个文件用于读写。文件指针（光标）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。

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。

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。

总结:

+ 增加同时读写的功能

a 追加（文件指针在文件尾）

以 w 打开文件时，文件不存在则创建，以 r 时， 不存在则报错

code 试一试

```

```

write & read 方法

```
fileObject.write(str)
str -- 要写入文件的字符串。
返回的是写入的字符长度。
fileObject.read(size);
size -- 从文件中读取的字节数，默认为 -1，表示读取整个文件。
返回从字符串中读取的字节。

<file>.readline(size = -1)  从文件读入一行内容，如果给出参数，读入该行前size长度的字符串或者字节流

<file>.readlines(hint = -1) 从文件中读入所有行，以每行为元素形成一个列表，如果给出参数，读入hint行

<file>.writeline(lines) 将一个元素全为字符串的列表写入文件

<file>.seek(offset[, whence])

- **offset** -- 开始的偏移量，也就是代表需要移动偏移的字节数，如果是负数表示从倒数第几位开始。
- **whence：**可选，默认值为 0。给 offset 定义一个参数，表示要从哪个位置开始偏移；0 代表从文件开头开始算起，1 代表从当前位置开始算起，2 代表从文件末尾算起。
```

with open() as f 的用法：

由于文件读写时有可能产生`IOError`，一旦出错，后面的`f.close()`就不会调用。所以，为了保证无论是否出错都能正确地关闭文件，我们可以使用`try ... finally`通过捕捉异常、处理异常来实现。

如果能保证文件打开没有异常的情况下，我们每次都这么写，实在太繁琐。所以，Python引入了`with`语句来自动帮我们调用`close()`方法。也就是说：文件读取的这个操作，只有在`with`语句内部才会生效，不会离开。

```
with open(文件名, 模式) as 文件对象:
    文件对象.方法()

with open('test.txt', 'r') as f:
    print(f.read())
```

str.join & str.strip & str.split

连接，移除，分割

str.join(sequence)
sequence -- 要连接的元素序列。
返回通过指定字符连接序列中元素后生成的**新字符串**。

str.strip([chars]);
chars -- 移除字符串头尾指定的字符序列。
返回移除字符串头尾指定的字符序列生成的**新字符串**。

str.split(str="", num=string.count(str))
str -- 分隔符，默认为所有的空字符，包括空格、换行(\n)、制表符(\t)等。
num -- 分割次数。默认为 -1, 即分隔所有。
返回分割后的**字符串列表**。

异常

try...except语句由两部分组成，一是try语句，二是except语句，try语句好比捕快，负责抓“犯人”，即捕获异常，except语句好比“法官”负责对“犯人”进行审判和处理，即对异常进行判断和处理

```
try:
    可能发生异常的代码
except
    发生异常时执行的代码
```

比如我们先让用户输入一个数字，然后用100除以这个数字，怎么写?

```
print('start')
num = int(input('请输入num'))
print(100//num)
print('over')

ValueError: invalid literal for int() with base 10: 'a'
ZeroDivisionError: integer division or modulo by zero
程序就凉了，不能往后运行了。
```

首先用户可能坑你，他不输入数字，怎么处理?

```
print('我不在捕获范围')
while True:
    try:
        num = int(input('请输入num'))
        break
    except:
        print('输入不合法，重新输入')
print(100//num)
print('over')
```

异常类型:

ValueError 传入无效参数

ZeroDivisonError 除数为0异常

接着捕获除数为0异常

```
print('我不在捕获范围')
while True:
    try:
        num = int(input('请输入num'))
        print(100 // num)
        break
    except ValueError:
        print('输入不合法，重新输入')
    except ZeroDivisionError:
        print('除数ye不能为0，请重新输入')

print('over')
```

最后一个，直接打印异常类型

```
print('我不在捕获范围')
while True:
    try:
        num = int(input('请输入num'))
        print(100 // num)
        break
    except Exception as e: #Exception为常规错误的基类
        print(e)

print('over')
```

异常处理结构中，try程序段中仅限第一个运行错误语句会被执行。

读取一维CSV，读取二维CSV