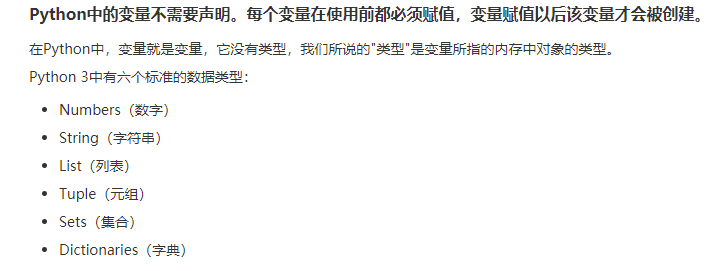
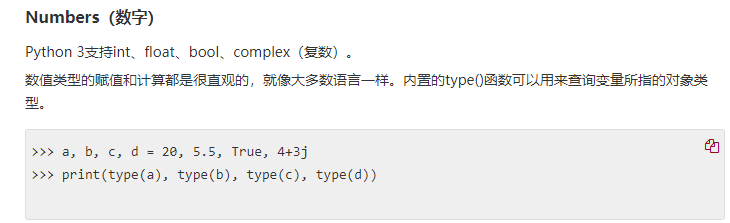
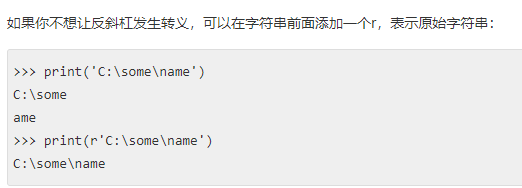
## 1 Python的变量和类型

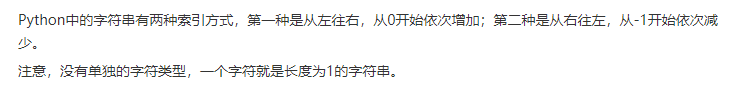


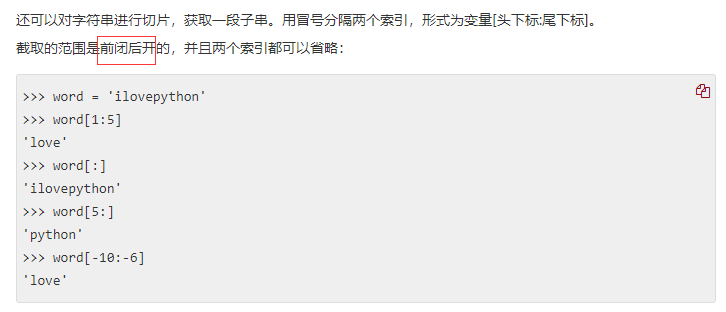
### Numbers类型

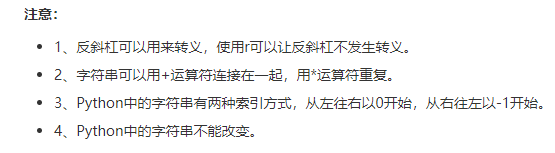


### String类型

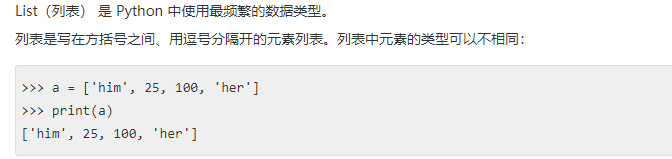


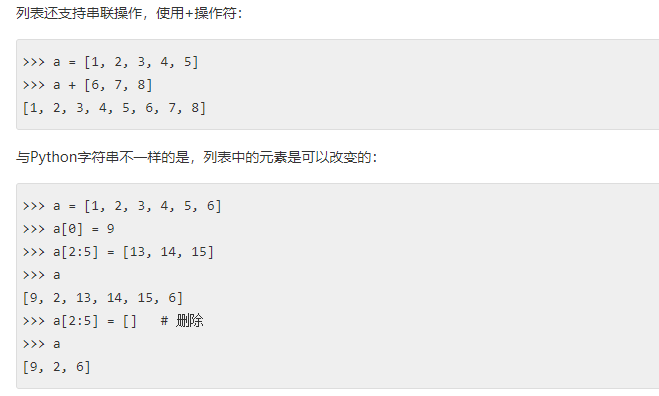




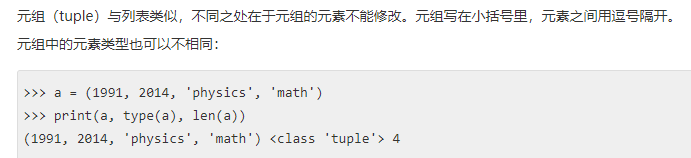


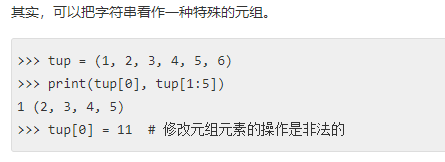
### List类型

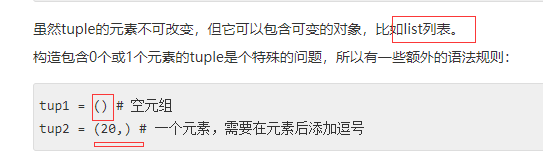




### Tuple类型

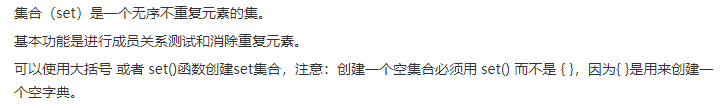


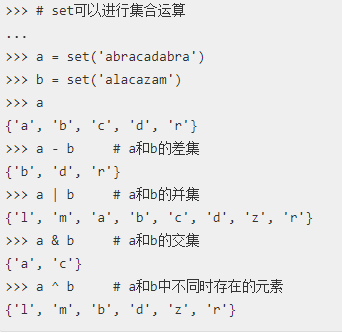




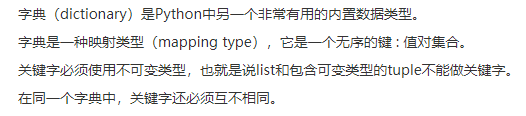


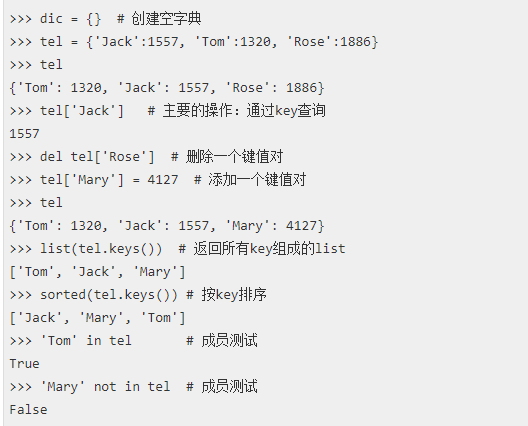
### Set类型

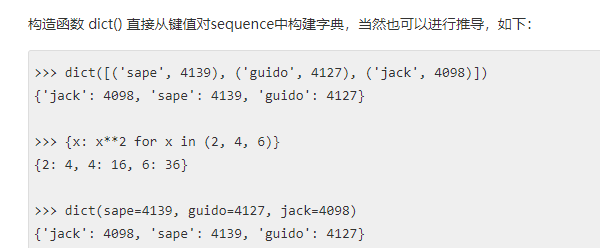




### Dictionaries类型







## 2 注释

单行：#

多行：’’’或者”””

## 3 运算符

主要注意\*\*(幂)和//(取整除)，两者还有赋值功能：\*\*=和//=；





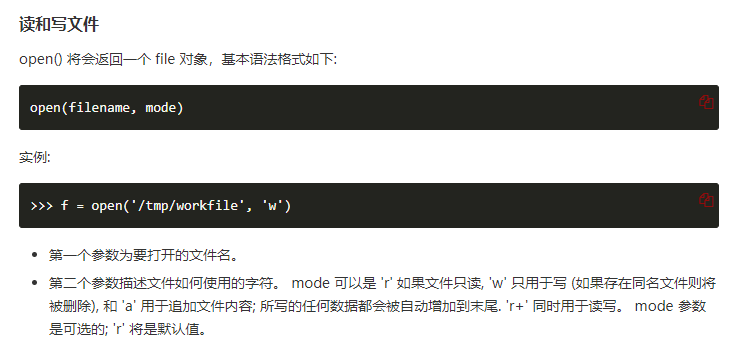


## 4 报错：IndexError: tuple index out of range

原因：参数数目不对，如：print('We are the {} who say {} who say "{}!"'.format('knights','Ni'))

只有两个参数传进去，但是我写错了，写了三个{}

## 5 文件对象的方法



### 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。

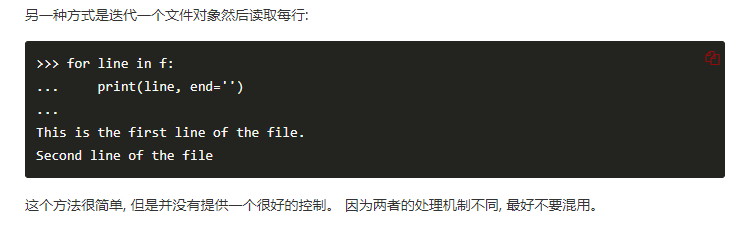
### f.readline()

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。

### 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。



### 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

### 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

<pre>

<p>

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:</p>

<pre>

>>> with open('/tmp/workfile', 'r') as f:

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

>>> f.closed

True

## 6 报错：io.UnsupportedOperation: can't do nonzero end-relative seeks

**原因：因为Pyhon3在文本文件中，没有使用b模式选项打开的文件，只允许从文件头开始计算相对位置，从文件尾计算时就会引发异常**

解决：使用’rb’打开文件就好了

## 7 序列化和反序列化

递归数据结构有指向原数据源的引用来表示，形式为<Recursion on typename with id=number>

## pickle 模块

python的pickle模块实现了基本的数据序列和反序列化。

通过pickle模块的序列化操作我们能够将程序中运行的对象信息保存到文件中去，永久存储。

通过pickle模块的反序列化操作，我们能够从文件中创建上一次程序保存的对象。

import pickle

# u表示unicode编码

data1 = {'a':[1,2.0,3,4+6j],'b':('string',u'Unicode string'),'c':None}

selfref\_list = [1,2,3]

selfref\_list.append(selfref\_list)

output = open('data.pkl','wb')

# Pickle dictionary using protocol 0.

pickle.dump(data1,output)

# Pickle the list using the highest protocol available.

pickle.dump(selfref\_list,output,-1)

output.close()

#使用pickle模块从文件中重构python对象

import pprint,pickle

with open("data.pkl",'rb') as pkl\_file:

data1 = pickle.load(pkl\_file)

pprint.pprint(data1)

data2 = pickle.load(pkl\_file)

pprint.pprint(data2)

# 递归数据结构有指向原数据源的引用来表示，形式为<Recursion on typename with id=number>

pkl\_file.closed

## 8 try except

try语句按照如下方式工作；

* 首先，执行try子句（在关键字try和关键字except之间的语句）
* 如果没有异常发生，忽略except子句，try子句执行后结束。
* 如果在执行try子句的过程中发生了异常，那么try子句余下的部分将被忽略。如果异常的类型和 except 之后的名称相符，那么对应的except子句将被执行。最后执行 try 语句之后的代码。
* 如果一个异常没有与任何的except匹配，那么这个异常将会传递给上层的try中。

一个 try 语句可能包含多个except子句，分别来处理不同的特定的异常。最多只有一个分支会被执行。

处理程序将只针对对应的try子句中的异常进行处理，而不是其他的 try 的处理程序中的异常。

一个except子句可以同时处理多个异常，这些异常将被放在一个括号里成为一个元组，例如:

except (RuntimeError, TypeError, NameError):

pass

最后一个except子句可以忽略异常的名称，它将被当作通配符使用。你可以使用这种方法打印一个错误信息，然后再次把异常抛出。

try except 语句还有一个可选的else子句，如果使用这个子句，那么必须放在所有的except子句之后。这个子句将在try子句没有发生任何异常的时候执行。

如果一个异常在 try 子句里（或者在 except 和 else 子句里）被抛出，而又没有任何的 except 把它截住，那么这个异常会在 finally 子句执行后再次被抛出。

关键词 with 语句就可以保证诸如文件之类的对象在使用完之后一定会正确的执行他的清理方法:

with open("myfile.txt") as f:

for line in f:

print(line, end="")

## 9 构造函数报错：TypeError: Complex() takes no arguments

检查你的   def \_\_init\_\_(self,   ,  ):函数是不是写成了   def \_int\_(self,   ,   ):

## 10 继承

需要注意圆括号中基类的顺序，若是基类中有相同的方法名，而在子类使用时未指定，python从左至右搜索 即方法在子类中未找到时，从左到右查找基类中是否包含方法。

BaseClassName（示例中的基类名）必须与派生类定义在一个作用域内。除了类，还可以用表达式，基类定义在另一个模块中时这一点非常有用:

class DerivedClassName(modname.BaseClassName):