Python实验报告

实验二 列表、元组、集合、字典的应用

学号：2016326603046 姓名：余泓锷

实验目的：

了解什么是列表、元组、集合与字典
了解列表、元组、集合与字典之间的差异
掌握列表、元组、集合与字典在Python的常用操作
掌握列表、元组、集合与字典的简单应用

实验内容：

实验题1：

以下是一系列书评,但是很多是灌水的,请你写一段代码,把灌水的书评尽可能去掉。

灌水书评一般会有一个特点：重复的字比较多，利用这个特点，把灌水书评去掉

'这是一本非常好的书，作者用心了',
'作者大大辛苦了',
'好书，感谢作者提供了这么多的好案例',
'书在运输的路上破损了，我好悲伤。。。',
'为啥我买的书上有菜汤。。。。',
'啊啊啊啊啊啊，我怎么才发现这么好的书啊，相见恨晚',
'书的质量有问题啊，怎么会开胶呢？？？？？？',
'好好好好好好好好好好好',
'好难啊看不懂好难啊看不懂好难啊看不懂',
'书的内容很充实',
'你的书上好多代码啊，不过想想也是，编程的书嘛，肯定代码多一些',
'书很不错!!一级棒!!买书就上当当，正版，价格又实惠，让人放心!!!',
'无意中来到你小铺就淘到心意的宝贝，心情不错!',
'送给朋友的、很不错',
'这是一本好书，讲解内容深入浅出又清晰明了，推荐给所有喜欢阅读的朋友同好们。',
'好好好，下一个看看',
'渣渣渣渣渣渣渣渣',
'没用没用没用没用',
'很好的书，五颗星'
解答：

In [2]:

```
text = ['这是一本非常好的书，作者用心了',
        '作者大大辛苦了',
        '好书，感谢作者提供了这么多的好案例',
        '书在运输的路上破损了，我好悲伤。。。',
        '为啥我买的书上有菜汤。。。。',
        '啊啊啊啊啊啊，我怎么才发现这么好的书啊，相见恨晚',
        '书的质量有问题啊，怎么会开胶呢？？？？？？',
        '好好好好好好好好好好好',
        '好难啊看不懂好难啊看不懂好难啊看不懂',
        '书的内容很充实',
        '你的书上好多代码啊，不过想想也是，编程的书嘛，肯定代码多一些',
        '书很不错!!一级棒!!买书就上当当，正版，价格又实惠，让人放心!!!',
        '无意中来到你小铺就淘到心意的宝贝，心情不错!',
        '送给朋友的、很不错',
        '这是一本好书，讲解内容深入浅出又清晰明了，推荐给所有喜欢阅读的朋友同好们。',
        '好好好，下一个看看',
        '渣渣渣渣渣渣渣渣',
        '没用没用没用没用',
        '很好的书，五颗星']
```

In [47]:

```
import re
hanzi = re.compile(r'[\u4e00-\u9fa5]')
for i in range(len(text)):
    #print(i)
    result = hanzi.findall(text[i])
    d = dict()
    for ch in result:
        d[ch] = d.get(ch,0) + 1
    flag = 0;
    for key,value in d.items():
        if value >= 2:
            flag += 1
    if flag < (int)(len(d)/3):
        print(text[i])
```

```
这是一本非常好的书，作者用心了
作者大大辛苦了
好书，感谢作者提供了这么多的好案例
书在运输的路上破损了，我好悲伤。。。
为啥我买的书上有菜汤。。。。
啊啊啊啊啊啊，我怎么才发现这么好的书啊，相见恨晚
书的质量有问题啊，怎么会开胶呢？？？？？？
书的内容很充实
你的书上好多代码啊，不过想想也是，编程的书嘛，肯定代码多一些
书很不错!!一级棒!!买书就上当当，正版，价格又实惠，让人放心!!!
无意中来到你小铺就淘到心意的宝贝，心情不错!
送给朋友的、很不错
这是一本好书，讲解内容深入浅出又清晰明了，推荐给所有喜欢阅读的朋友同好们。
很好的书，五颗星
```

实验题2：

以下是一段文字，统计出现最多的英文单词

Python Success Stories

Background

Industrial Light & Magic (ILM) was started in 1975 by filmmaker George Lucas, in order to create the special effects for the original Star Wars film. Since then, ILM has grown into a visual effects powerhouse that has contributed not just to the entire Star Wars series, but also to films as diverse as Forrest Gump, Jurassic Park, Who Framed Roger Rabbit, Raiders of the Lost Ark, and Terminator 2. ILM has won numerous Academy Awards for Best Visual Effects, not to mention a string of Clio awards for its work on television advertisements.

While much of ILM's early work was done with miniature models and motion controlled cameras, ILM has long been on the bleeding edge of computer generated visual effects. Its computer graphics division dates back to 1979, and its first CG production was the 1982 Genesis sequence from Star Trek II: The Wrath of Khan.

In the early days, ILM was involved with the creation of custom computer graphics hardware and software for scanning, modeling, rendering, and compositing (the process of joining rendered and scanned images together). Some of these systems made significant advances in areas such as morphing and simulating muscles and hair.

Naturally, as time went by many of the early innovations at ILM made it into the commercial realm, but the company's position on the cutting edge of visual effects technology continues to rely on an ever-changing combination of custom in-house technologies and commercial products.

Today, ILM runs a batch processing environment capable of modeling, rendering and compositing tens of thousands of motion picture frames per day. Thousands of machines running Linux, IRIX, Compaq Tru64, OS X, Solaris, and Windows join together to provide a production pipeline that is used by approximately eight hundred users daily, many of whom write or modify code that controls every step of the production process. In this context, hundreds of commercial and in-house software components are combined to create and process each frame of computer-generated or enhanced film.

Making all this work, and keeping it working, requires a certain degree of technical wizardry, as well as a tool set that is up to the task of integrating diverse and frequently changing systems.

Enter Python Back in 1996, in the 101 Dalmation days, ILM was exclusively an SGI IRIX shop, and the production pipeline was controlled by Unix shell scripting. At that time, ILM was producing 15-30 shots per show, typically only a small part of each feature length film to which they were contributing.

Looking ahead towards more CG-intensive films, ILM staff began to search for ways to control an increasingly complex and compute-intensive production process.

It was around this time that Python version 1.4 came out, and Python was coming into its own as a powerful yet simple language that could be used to replace Unix shell scripting. Python was evaluated, compared to other technologies available at the time (such as Tcl and Perl), and chosen as an easier to learn and use language with which to incrementally replace older scripts.

At ILM, speed of development is key, and Python was a faster way to code (and re-code) the programs that controlled the production pipeline.

Python Streamlines Production But Python was not designed just as a replacement for shell scripting and, as it turns out, Python enabled much more for ILM than just process control.

Unlike Unix shell scripting, Python can be embedded whole as a scripting language within a larger software system. In this case, Python code can invoke specific functions of that system, even if those functions are written in C or C++. And C and C++ code can easily make calls back into Python code as well.

Using this capability, ILM integrated Python into custom applications written in C or C++, such as ILM's in-house lighting tool, which is used to place light sources into a 3D scene and to facilitate the writing, generation, and previewing of shaders and materials used on CG elements. It is the lighting tool that is ultimately responsible for

writing the 3D scene out to a format that a renderer can interpret and render.

At the same time, more and more components, such as those responsible for ILM's many custom file formats and data structures, were re-wrapped as Python extension modules

As Python was used more widely, extending and customizing in-house software became a lot easier. By writing in Python, users could recombine wrapped software components and extend or enhance standard CG applications needed for each new image production run. This let ILM staff to do exactly what a production needed at any given time, whether that meant allowing for a specific look for an entire show, or just a single CG character or element.

As it turned out, even some of ILM's non-technical users were able to learn enough Python to develop simple plug-ins and to create and modify production control scripts along side the technical users.

Python Unifies the Toolset Encouraged by its successes in batch process control and in scripting applications and software components, ILM started to use Python in other applications as well.

Python is now used for tracking and auditing functionality within the production pipeline, where an Oracle database keeps track of the hundreds of thousands of images that are created and processed for each film. DCOracle2, one of the Oracle integration libraries available for Python, has performed well in this task and is now in use on Linux, IRIX, Tru64, and Solaris.

Python is also used to develop the CG artist's interface to ILM's asset management system. Designed to be modular, this tool has been enhanced by a large group of developers and non-developers alike to extend well beyond its original mandate. The application is now used not only to manage CG assets and elements, but also in daily shot review, as a network-based whiteboard, as an instant messenger, and even allows an occasional game of chess.

As Python was applied in more ways, it slowly crowded out a plethora of competing technologies for shell scripting and batch control, embedded scripting, component software development, database application development, and so forth. Python's versatility ultimately simplified the developers' toolset and reduced the number of technologies that needed to be deployed to ILM's thousands of production computers. This new, simpler toolset translated directly into an easier to manage and more reliable development and production process.

Hardware Costs Reduced Although chosen initially for its ease of use and integration capabilities, Python's portability to many other operating systems eventually became one of its key strengths.

Once Python was in use, it made the production control system portable. This gave ILM additional freedom in making hardware technology choices, including a large-scale introduction of commodity PC hardware and Linux, a move that has saved the company substantial amounts of money in recent years.

Source Code Access Important After having used Python intensively for six years, ILM has yet to run into significant bugs or portability issues with the language. As a result, ILM has since Python 1.5 been able to rely on stock distributions in unmodified form.

However, availability of source code for the language acts as an important insurance policy should problems arise in the future, or should custom extensions or improvements become necessary. Without this, ILM could never have bought into Python so heavily for its mission-critical production process.

One case where access to source has already been beneficial was in ILM's continued use of Python 1.4, which is generally considered obsolete. Because the production facility is under continuous use, upgrading systems to new Python versions would result in significant disruption of the production process.

Instead, ILM installs new systems with newer versions of Python but maintains older systems only so they can run the same scripts as the newer systems. Supporting this mix has relied on access to the Python sources in order to back-port some changes found in newer Python versions, and to reimplement portions of newer support libraries under older versions of Python. ILM is currently running a mix of Python 1.4, 1.5, and 2.1.

Python Tested by Time The visual effects industry is intensely competitive. To stay on top of the heap, ILM continuously reviews its production methods and evaluates new technologies as they become available.

Since its adoption in 1996, the use of Python has also been reviewed numerous times. Each time, ILM failed to find a more compelling solution. Python's unique mix of simplicity and power continues to be the best available choice for controlling ILM's complex and changing computing environment.

About the Author Tim Fortenberry joined Industrial Light & Magic in 1999 as an intern. Later that same year he began to work full time in the Resources department. He worked as a scripts/tools programmer. Shortly after, Fortenberry joined the Research and Development department. He is one of the founding members of the Pipeline and TD Tools groups that helped bridge the gap between artists and technology.

As an engineer, Fortenberry is responsible for developing and maintaining the myriad of applications used for rendering and pipline control flow of images at ILM. Prior to joining ILM, Fortenberry worked as a Linux systems administrator for VA Linux Systems.

Originally from Southern California, Fortenberry received his Bachelor of Arts degree from the University of California at Berkeley in Anthropology with an emphasis in Archaeology.
解答：

In [48]:

```
article = '''Python Success Stories

Background

Industrial Light & Magic (ILM) was started in 1975 by filmmaker George Lucas, in order to create the

While much of ILM's early work was done with miniature models and motion controlled cameras, ILM has

In the early days, ILM was involved with the creation of custom computer graphics hardware and softw

Naturally, as time went by many of the early innovations at ILM made it into the commercial realm, b

Today, ILM runs a batch processing environment capable of modeling, rendering and compositing tens o

Making all this work, and keeping it working, requires a certain degree of technical wizardry, as we

Enter Python Back in 1996, in the 101 Dalmation days, ILM was exclusively an SGI IRIX shop, and the

Looking ahead towards more CG-intensive films, ILM staff began to search for ways to control an incr

It was around this time that Python version 1.4 came out, and Python was coming into its own as a po

At ILM, speed of development is key, and Python was a faster way to code (and re-code) the programs

Python Streamlines Production But Python was not designed just as a replacement for shell scripting

Unlike Unix shell scripting, Python can be embedded whole as a scripting language within a larger so

Using this capability, ILM integrated Python into custom applications written in C or C++, such as I

At the same time, more and more components, such as those responsible for ILM's many custom file for

As Python was used more widely, extending and customizing in-house software became a lot easier. By

As it turned out, even some of ILM's non-technical users were able to learn enough Python to develop

Python Unifies the Toolset Encouraged by its successes in batch process control and in scripting app

Python is now used for tracking and auditing functionality within the production pipeline, where an

Python is also used to develop the CG artist's interface to ILM's asset management system. Designed

As Python was applied in more ways, it slowly crowded out a plethora of competing technologies for s

Hardware Costs Reduced Although chosen initially for its ease of use and integration capabilities, P

Once Python was in use, it made the production control system portable. This gave ILM additional fre

Source Code Access Important After having used Python intensively for six years, ILM has yet to run

However, availability of source code for the language acts as an important insurance policy should p

One case where access to source has already been beneficial was in ILM's continued use of Python 1.4

Instead, ILM installs new systems with newer versions of Python but maintains older systems only so

Python Tested by Time The visual effects industry is intensely competitive. To stay on top of the he

Since its adoption in 1996, the use of Python has also been reviewed numerous times. Each time, ILM
```

```
About the Author Tim Fortenberry joined Industrial Light & Magic in 1999 as an intern. Later that sa

As an engineer, Fortenberry is responsible for developing and maintaining the myriad of applications

Originally from Southern California, Fortenberry received his Bachelor of Arts degree from the Unive
```

In [73]:

```
#print(article)
import re
str1 = re.sub(r'[\n\t ]',',',article)
words = str1.split(',')
while '' in words:
    words.remove('')
dic = dict()
for word in words:
    dic[word] = dic.get(word,0) + 1
max_value = dic[max(dic,key=dic.get)]
for key,value in dic.items():
    if value == max_value:
        print(key,value)
```

```
and 60
```