



程序设计 (Python)

文件与异常

主讲：数据与目标工程学院 胡瑞娟 副教授

【第7讲作业】编写程序实现**词频统计**，具体要求：

实现以下功能：

- ✧ 接收用户输入的一段文本（包含多个单词）
- ✧ 将文本中的单词按照某种方式分离并保存，以便后续处理
- ✧ 找出文本中所有不同的单词（排除重复项）
- ✧ 统计每个不同单词在文本中出现的次数

【提示】

思考使用合适的数据结构来分别实现以下功能：

- ✧ 存储原始文本中的所有单词（允许重复）
- ✧ 记录文本中出现过的不同单词（不允许重复）
- ✧ 存储每个不同单词与其出现次数的对应关系

```
# 1. 定义待统计的文本（可替换为自己的文本内容）
text = """Transformer Architecture: The foundation of modern LLMs """

# 2. 文本预处理：转为小写、去除标点符号
punctuations = "。、：；？！‘’‘‘‘‘\n"
processed_text = text.lower() # 统一转为小写，避免大小写差异影响统计
for p in punctuations:
    processed_text = processed_text.replace(p, " ")

# 3. 分词：按空格分割成单词列表，并过滤空字符串
words = processed_text.split(" ")
words = [word for word in words if word != ""] # 去掉分割后产生的空元素

# 4. 不重复单词列表
word_list=[]
for word in words:
    if word not in word_list:
        word_list.append(word)

# 5. 统计词频
word_freq = {}
for word in words:
    if word in word_freq:
        word_freq[word] += 1 # 单词已存在，计数+1
    else:
        word_freq[word] = 1 # 单词首次出现，初始化为1

# 6. 输出词频统计结果
print("所有单词列表: ",words)
print("不重复单词列表: ",word_list)
print("词频统计结果: ")
for word, count in word_freq.items():
    print(f"{word} {count}次")
```

text_pro()

参数：字符串

返回值：列表

get_unique_words()

参数：列表

返回值：列表

#函数：预处理与分词

```
def text_pro(text):
    text_lower = text.lower()
    punctuations = ['.', ',', '!', '?', ';', ':', '“', "”", '(', ')', '[', ']', '{', '}']
    for p in punctuations:
        text_lower = text_lower.replace(p, ' ')
    words = text_lower.split()
return words
```

#函数：生成不重复单词列表

```
def get_unique_words(ls):
    word_list = []
    for word in ls:
        if word not in word_list:
            word_list.append(word)
return word_list
```

【案例背景】在人工智能飞速发展的今天，各类技术文献的数量呈指数级增长。无论是阅读最新的AI论文、学习前沿的技术文档，还是分析国际科技公司的研究报告，我们都需要**处理大量的技术文本**。假设你是一位AI领域的研究人员，现需要快速分析关于"大模型LLM"相关的文章，提取关键术语构建一个领域术语库。手动处理文本既耗时又容易出错，这时就需要设计一个**词频统计器**，它不仅能帮你快速处理技术文献，还能为你后续的词云分析、主题建模等高级任务打下基础。

Large Language Models (LLMs) Technical Overview

Definition and Architecture:

Large Language Models are deep learning algorithms that can

Key Components:

- Transformer Architecture: The foundation of modern LLMs
- Attention Mechanisms: Allow models to focus on relevant parts
- Embedding Layers: Convert tokens to vector representations
- Feed-Forward Networks: Process information at each layer
- Layer Normalization: Stabilize training process

1.从文本中提取不重复单词



large
language
models
llms
technical
overview
definition
and
architecture
are
deep

2.去除停用词后统计词频



large 3
language 4
models 9
llms 5
technical 2
overview 1
definition 1
architecture 3
deep 1
learning 2
algorithms 1

【问题1】如何从文本中提取不重复单词。



read

数据驱动思维

生成不重复单词

write



读文件

写文件

从文件中读取出内容

处理

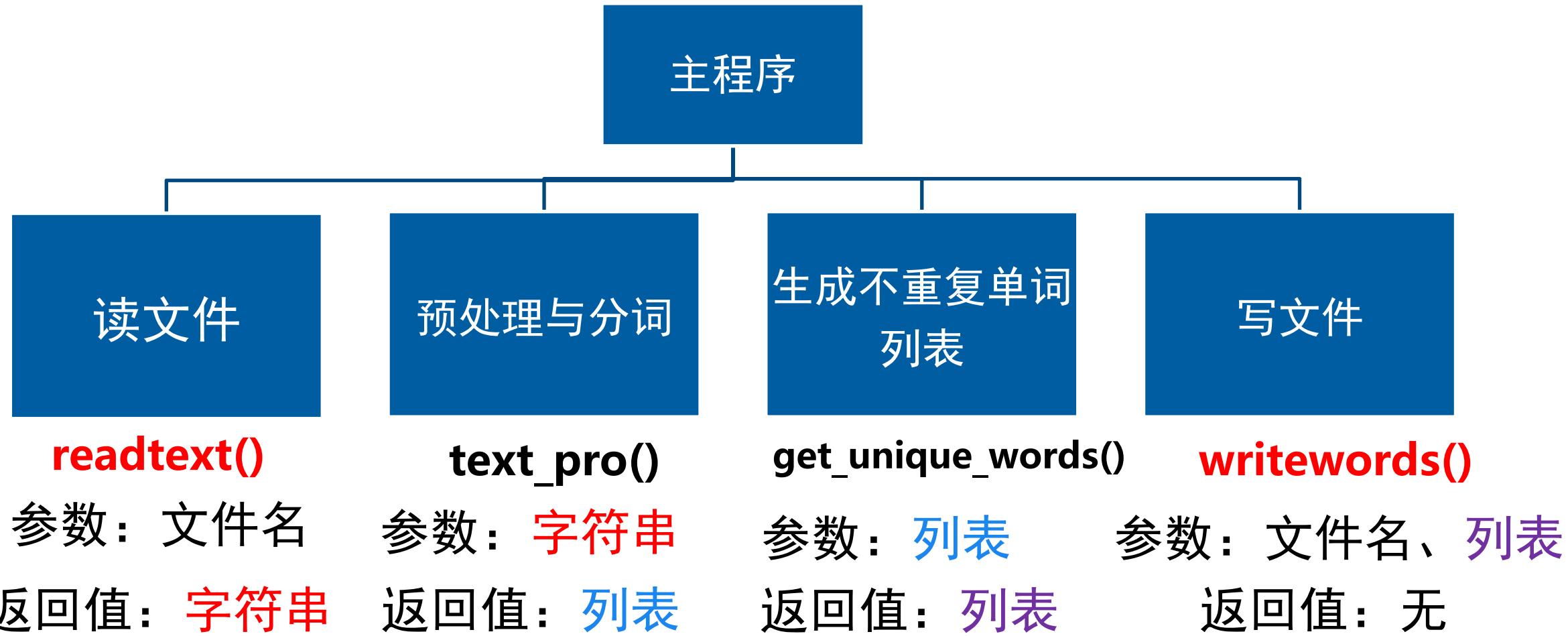
将结果写入到文件

I

P

O

【问题1】如何从文本中提取不重复单词。



01 | 文件概述

02 | 文件的操作

打开、读、写入、关闭

03 | 异常



01



文件

所有的数据和程序都是以()形式存在磁盘上

文件夹

文件



记录

信息

文件是数据的抽象和集合

- 文件是存储在**辅助存储器上的数据序列**
- 文件是**数据存储的一种形式**
- 本质都是**二进制文件（物理存储结构）**
- 文件**展现形态（逻辑存储结构）：文本文件和二进制文件**



➤ 从Python角度，文件为什么要分为**文本文件**和**二进制文件**？

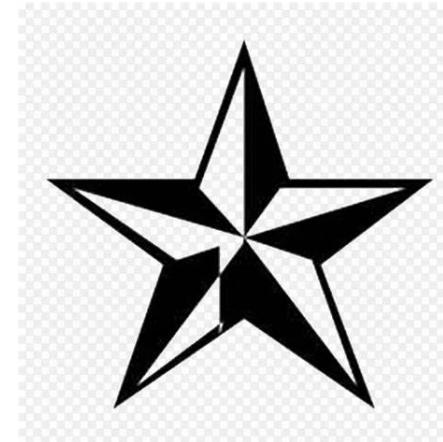
文本文件



大多数情况更关心**文件中的字符**

读：字节→按照**指定编码解码**→字符串
写：字符串→按照**指定编码编码**→字节

二进制文件

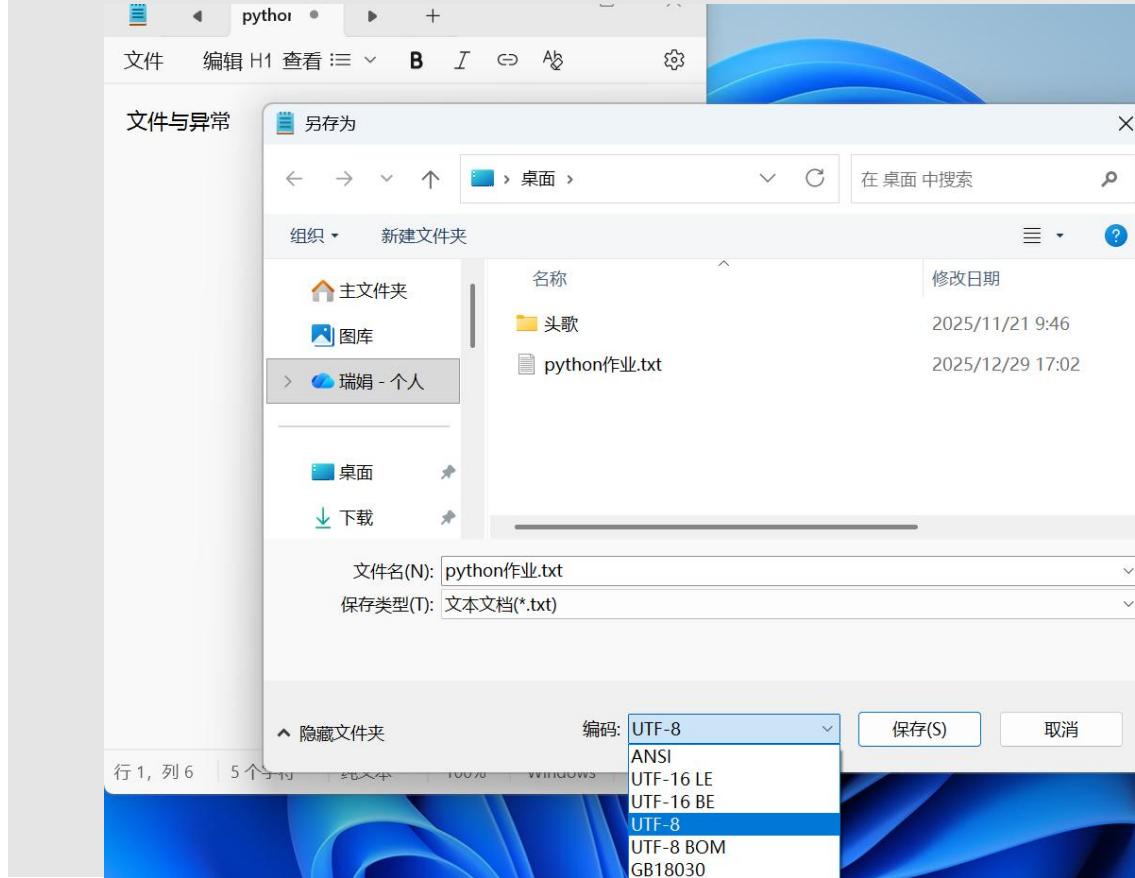


1	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	1	0
0	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	1	0	0	1	0	0	0
0	1	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	1

按照**字节流处理**文件

读：字节→字节流对象
写：字节流对象→字节

文本文件



二进制文件

- 由比特0和1组成，以字节流进行存储，
没有统一字符编码
- 非字符但特定格式（文件格式）形成的文件，不能用记事本等直接编辑和
阅读，只能使用专门的软件打开
- 适用于：.png文件、.avi文件等多媒体文件、office文档、exe文件等

本质区别：文本文件有统一的字符编码形式，而二进制文件没有。

下列选项错误的是（ ）。

- A 文本文件由单一编码的字符组成
- B 文本文件是存储在磁盘上的长字符序列
- C 文本文件容易阅读和展示
- D 文本文件可由多种编码的字符组成

➤ 从Python的角度来看，以下几个文件分别是**文本文件**还是**二进制文件**？



2025两会.txt



阶段小测2.1：中文词频统计（保存csv）.py



字典特点1.png

02



文件的操作

(1) 打开文件

内置函数**open()**用于打开文件，该方法的声明如下：

```
open(file, mode='r', encoding="编码")
```

【参数】

- **file**: 文件的路径。
- **mode**: 设置文件的打开模式，取值有**r**、**w**、**a**。
- **encoding**: 设置访问文件的编码格式。
- 若**open()**函数调用成功，返回一个**文件对象**。

- **r**: 以**只读**方式打开文件
(**mode**参数的默认值)。
- **w**: 以**覆盖写**方式打开文件。
- **x**: 以**创建**的方式打开文件。
- **a**: 以**追加**方式打开文件。
- **b**: 以**二进制**形式打开文件。
- **+**: 以**更新**的方式打开文件
(可读可写)。

四选一
二选一

文件的打开模式	描述
'r'	只读模式, 默认值 , 如果文件不存在, 返回FileNotFoundException
'w'	覆盖写模式, 文件不存在则创建, 存在则完全覆盖
'x'	创建写模式, 文件不存在则创建, 存在则返回FileExistsError
'a'	追加写模式, 文件不存在则创建, 存在则在文件最后追加内容
'b'	二进制文件模式
't'	文本文件模式, 默认值
'+'	与r/w/x/a一同使用, 在原功能基础上增加同时读写功能

➤ 文件打开模式练习

```
f = open("f.txt")
```

```
f = open("f.txt", "rt")
```

```
f = open("f.txt", "w")
```

```
f = open("f.txt", "a+")
```

```
f = open("f.txt", "x")
```

```
f = open("f.txt", "b")
```

```
f = open("f.txt", "wb")
```

文本/二进制形式、读写方式

- **文本形式、只读模式、默认值**
- **文本形式、只读模式、同默认值**
- **文本形式、覆盖写模式**
- **文本形式、追加写模式+读文件**
- **文本形式、创建写模式**
- **错误，二进制形式、只读模式，b**
- 不能单独用，必须和r/w/a/x一起**
- **二进制形式、覆盖写模式**

使用'w'模式打开已存在的文件会发生什么？

- A 追加内容
- B 文件内容保持不变
- C 文件被清空
- D 抛出错误

(2) 关闭文件

方法一：close()方法

- close()方法是文件对象的内置方法。

file.close()

示例

为什么要及时关闭文件？

- 计算机中可打开的文件数量是有限
- 打开的文件占用系统资源
- 若程序因异常关闭，可能产生数据丢失
- 频繁打开和关闭文件会影响程序的执行效率，及时关闭可以减少系统的开销
- 程序员有责任确保资源得到适当的管理

(2) 关闭文件

Python可通过**close()**方法关闭文件，也可以使用**with语句**实现文件的自动关闭。

方法二：with语句

- 使用保留字**with**和**as**以及**open()**函数，可以简化文件打开，无需显示关闭。
当程序退出**with**区域代码执行时，将自动关闭所打开的文件f。

```
with open("11m.txt","rt",encoding="utf-8") as f:  
    text=f.read()  
    print(text)
```

示例

(3) 读取文件内容

文件作为一个对象，`<a>. `方式进行操作

操作方法	描述
<code><f>.read(size=-1)</code>	<p>读入全部内容，如果给出参数，读入前size字符长度 <code>>>>s = f.read(2)</code>#返回值为字符串 中国</p>
<code><f>.readline(size=-1)</code>	<p>读入一行内容，如果给出参数，读入该行前size字符长度 <code>>>>s = f.readline()</code> #返回值为字符串 中国是一个伟大的国家！</p>
<code><f>.readlines(hint=-1)</code>	<p>读入文件所有行，以每行为元素形成列表 如果给出参数，读入前hint行 <code>>>>s = f.readlines()</code> ['中国是一个伟大的国家！\n', '历史悠久...\n']</p>

【举一反三】读取1.txt文件

```
f = open("1.txt","rt",encoding="utf-8")
text = f.read()#读全部内容
print(text)
f.close()
```

文本语料1-1
文本语料1-2
文本语料1-3
文本语料1-4
文本语料1-5

```
f = open("1.txt","rt",encoding="utf-8")
text1 = f.readline()#读一行内容
text2 = f.readline()
print(text2)
f.close()
```

文本语料1-2

【举一反三】读取1.txt文件

```
f = open("1.txt","rt",encoding="utf-8")
text = f.readlines()#以列表返回多行内容
print(text)
f.close()
```

如何读取第4行信息？



['文本语料1-1\n', '文本语料1-2\n', '文本语料1-3\n', '文本语料1-4\n', '文本语料1-5']



f.readlines() 返回一个**列表**，每个元素都是**字符串**，元素中有**换行符**

【举一反三】读取1.txt文件，逐行遍历

```
f = open("1.txt", "rt", encoding="utf-8")
for line in f.readlines(): #一次读入，分行处理
    print(line, end="")
f.close()
```



文本语料1-1

文本语料1-2

文本语料1-3

文本语料1-4

文本语料1-5

```
f = open("1.txt", "rt", encoding="utf-8")
for line in f: #分行读入，分行处理
    print(line, end="")
f.close()
```



当文件比较大时，一次读入会占用很多内存，建议使用分行读入到内存。

文本语料1-1

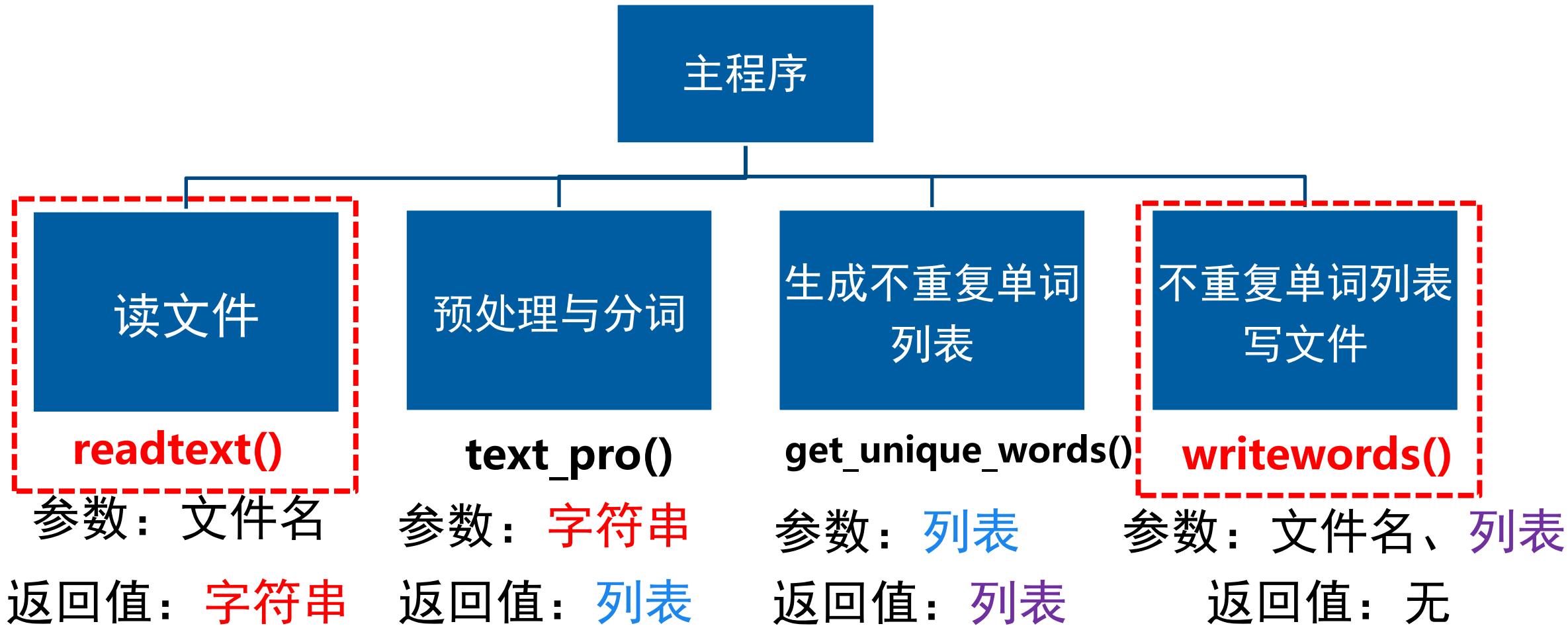
文本语料1-2

文本语料1-3

文本语料1-4

文本语料1-5

【案例问题1解决】如何从文本中提取不重复单词。



【案例问题1解决】如何从文本中提取不重复单词。

➤ 读文件函数**readtext()**

```
def readtext(filename):
    with open("11m.txt","rt",encoding="utf-8") as f:
        s=f.read()
    return s
```

(4) 文件的写入

操作方法	描述
<f>.write(s)	向文件写入一个 字符串 或字节流 <code>>>>f.write("中国是一个伟大的国家!")</code>
<f>.writelines(lines)	将一个 元素全为字符串的列表 写入文件 <code>>>>ls = ["中国", "法国", "美国"]</code> <code>>>>f.writelines(ls)</code> <code>中国 法国 美国</code>
<f>.seek(offset)	改变当前文件操作指针的位置，offset含义如下： 0 – 文件开头； 1 – 当前位置； 2 – 文件结尾 <code>>>>f.seek(0) #回到文件开头</code>

【案例问题1解决】如何从文本中提取不重复单词。

➤ 写入文件writewords()

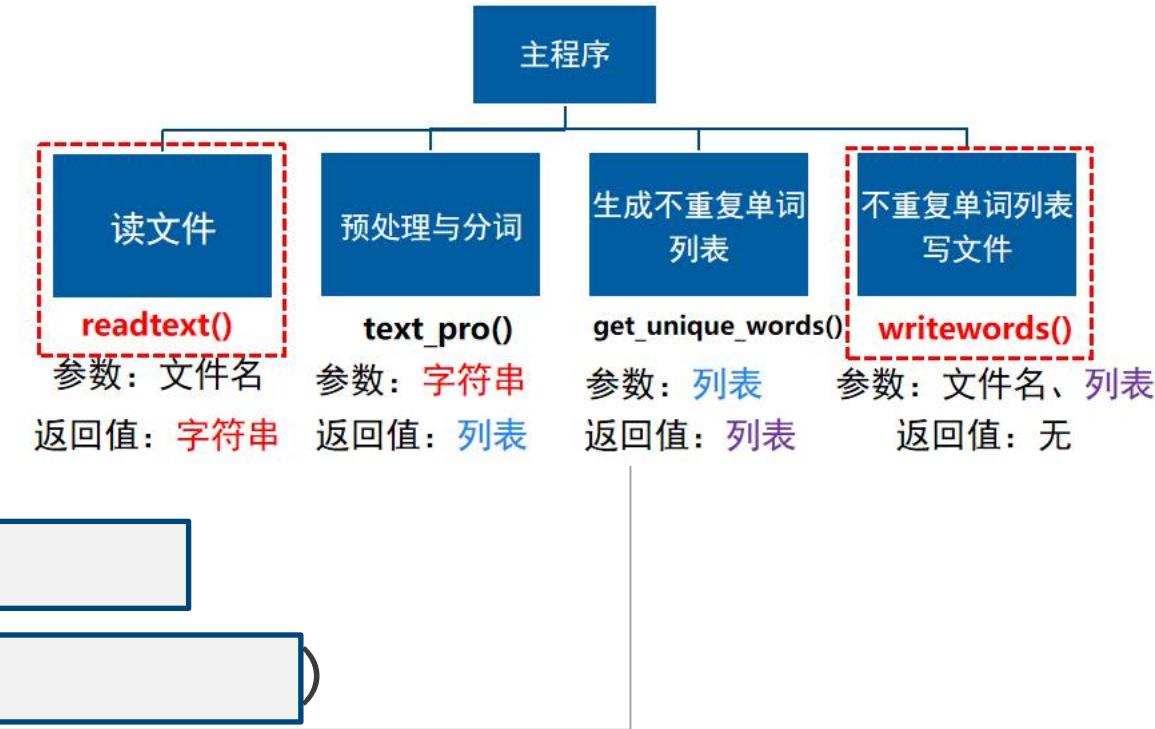
```
#函数：把单词列表写入到文件

def writewords(filename,wordlist):
    nwordlist=[word+"\n" for word in wordlist]
    with open(filename,"w",encoding="utf-8") as f:
        f.writelines(nwordlist)
```

【案例问题1解决】如何从文本中提取不重复单词。

➤ 函数的调用

```
s=   
words=   
unique_words=   
writewords(  )
```



思考：排序后的单词写入到txt文件？

写入文件前插入一句：

```
unique_words.sort() #默认升序，两个参数key, reverse
```

2. 文件的操作

【问题2解决】如何从文本中提取不重复单词，**统计词频**（先不去除停用词），并将结果保存在文件（l1m_wordcount1.txt）中（一词占一行）。

Large Language Models (LLMs) Technical Overview

Definition and Architecture:

Large Language Models are deep learning algorithms that can

Key Components:

- Transformer Architecture: The foundation of modern LLMs
- Attention Mechanisms: Allow models to focus on relevant parts
- Embedding Layers: Convert tokens to vector representations
- Feed-Forward Networks: Process information at each layer
- Layer Normalization: Stabilize training process

1. 从文本中提取不重复单词



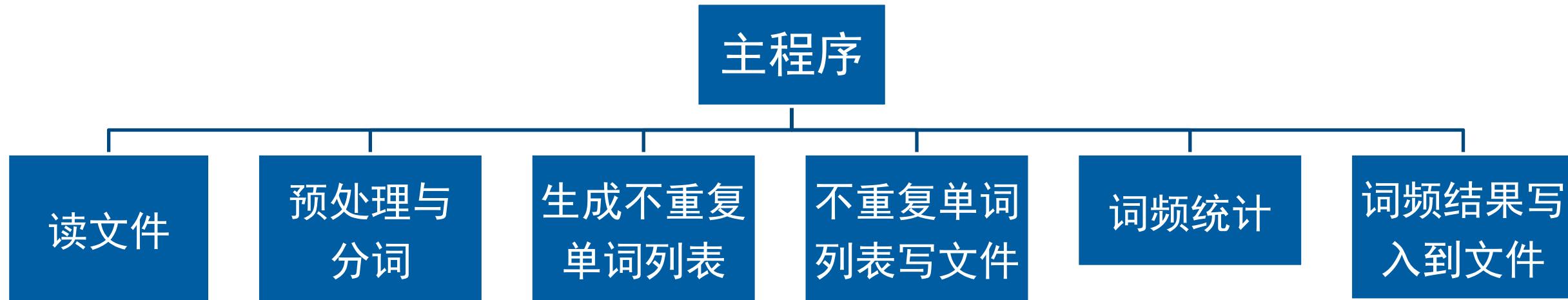
large
language
models
llms
technical
overview
definition
and
architecture
are
deep

2. 去除停用词后统计词频



large 3
language 4
models 9
llms 5
technical 2
overview 1
definition 1
architecture 3
deep 1
learning 2
algorithms 1

【案例问题2分析】词频统计——模块化设计



参数	文件名	字符串	列表	文件名/列表	列表	文件名/字典
返回值	字符串	列表	列表	无	字典	无



【案例问题2分析】词频统计

```
#新增函数：统计词频，得到词频字典
def count_words(ls):
    counts = {}
    for word in ls:
        #如果不在字典中则添加
        if word not in counts:
            counts[word] += 1
        else:
            counts[word]=1
    return counts
```

【案例问题2解决】词频统计

思路：1. 字典写入txt文件

字典

counts = {'fellow': 4, 'deputies': 3, 'on': 89,

一部分词
频结果

问题求解

打开文件（模式：**新建写**）

#字典写入文件

遍历字典的键值对

将一个词、空格、词频、\n链接成一个字符串，写入到文件中

关闭文件

【案例问题2解决】词频统计

思路：字典写入txt文件writewordfrq

新建写

```
#函数：词频结果写入到文件
def writewordfrq(filename, counts):
    with open(filename, "w", encoding="utf-8") as f:
        for word, count in counts.items():
            f.write(word + " " + str(count) + "\n")
```

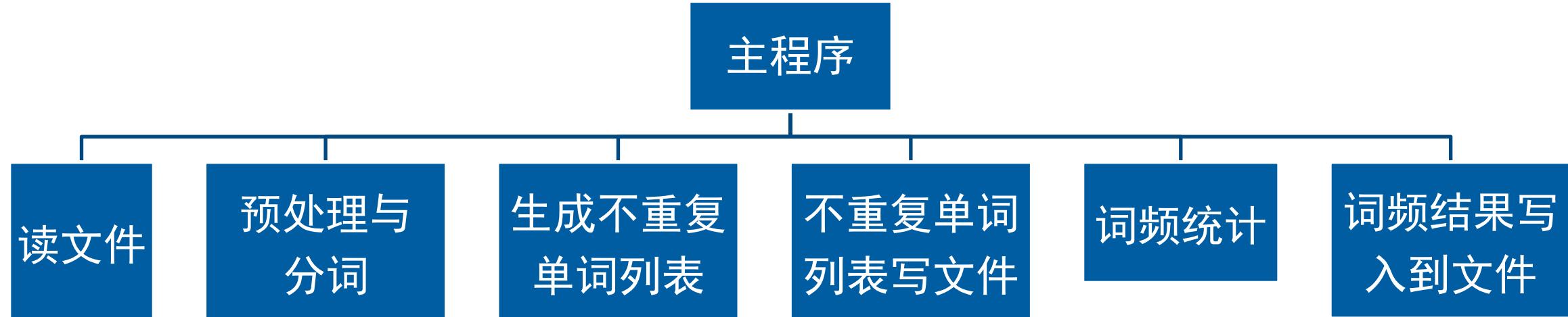
每个单词和词频
构造成字符串

【案例问题2解决】词频统计

```
#函数调用
s=readtext("11m.txt")
words=text_pro(s)
unique_words=get_unique_words(words)
unique_words.sort()
writewords("11m_unique.txt",unique_words)
words_dict=count_words(words)
writewordfrq("11m_wordcount1.txt",words_dict)
```

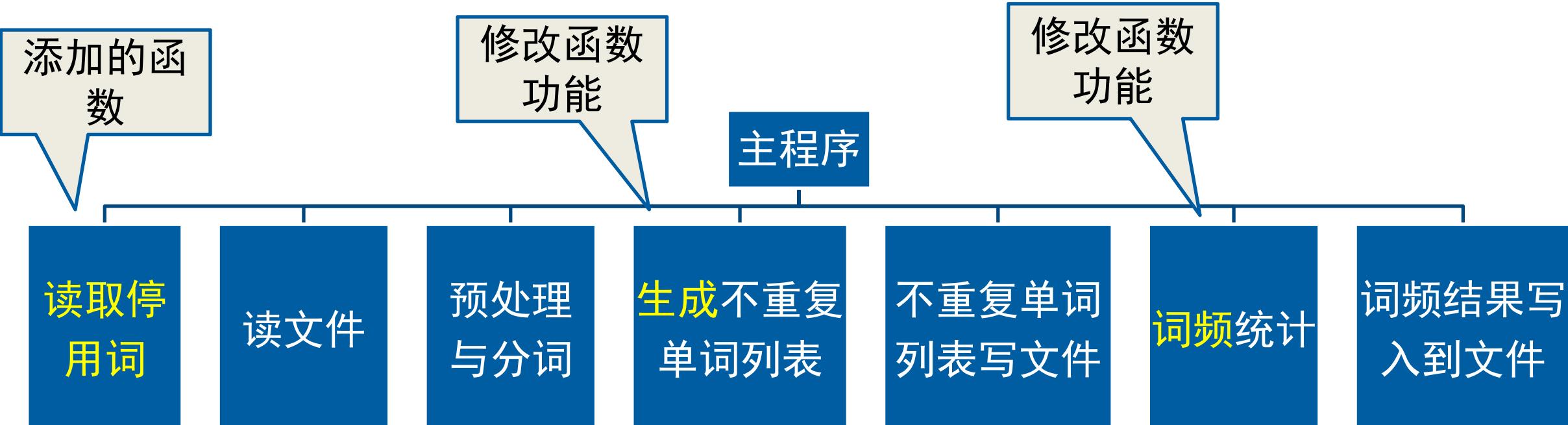
【问题3分析】去除停用词

- 在问题2的基础上，利用**停用词文件**，统计**不含停用词的不重复单词和词频结果**并写入到文件。



思考：处理停用词的思路？
添加或者修改哪些函数？

【问题3分析】去除停用词



添加修改后的函数划分

【问题3分析】去除停用词

(1) 从文件中读取停用词

- 思路：停用词保存为哪种数据类型？
- 问题：读文件——列表
- 思考：使用哪种文件读方法？



f. read()

f. readline()

f. readlines()



【问题3解决】

(1) 从文件中读取停用词



f. read()
f. readline()
f. readlines()

#新增函数：读取停用词

```
def readstopwords(filename):
    with open(filename, "r", encoding="utf-8") as f:
        stop_words=f.read().split()
    return stop_words
```

【问题3解决】

(2) 利用停用词列表修改生成不重复单词和词频统计的函数

#修改函数：生成去除停用词不重复单词列表

```
def get_unique_words(ls,stopwords):
    word_list=[]
    for word in ls:
        if word not in word_list and word not in stopwords:
            word_list.append(word)
    return word_list
```

【问题3解决】

(2) 利用停用词列表修改生成不重复单词和词频统计的函数

#修改函数：统计词频，得到不含停用词的词频字典

```
def count_words(ls, stopwords):
    counts = {}
    for word in ls:
        #如果不在停用词中则添加到字典中
        if word not in stopwords:
            counts[word] = counts.get(word, 0) + 1 #get方法：不存在则返回0，存在则返回原值
    return counts
```

【问题3解决】

```
#函数调用
s=readtext("11m.txt")
stopwords=readstoptext("en_stopwords.txt")
words=text_pro(s)
unique_words=get_unique_words(words,stopwords)
writewords("11m(去除停用词).txt",unique_words)
counts=count_words(words,stopwords)
writewordfrq("11m词频统计(去除停用词).txt",counts)
```

如果停用词表en_stopwords.txt不存在，会发生什么？

Traceback (most recent call last):

```
  File "D:\教学\2025-2026秋\程序设计Python\testcode\file_wordlist.py", line 169, in <module>
    stopwords = readstoptext("en_stopwords.txt")
  File "D:\教学\2025-2026秋\程序设计Python\testcode\file_wordlist.py", line 8, in readstoptext
    with open(filename, "r", encoding="utf8") as f:
FileNotFoundError: [Errno 2] No such file or directory: 'en_stopwords.txt'
```

—— 实际输出 ——

展示原始输出

```
Traceback (most recent call last):
  File "step19/recursive.py", line 8, in <module>
    ddd=ns
NameError: name 'ns' is not defined
```

```
Traceback (most recent call last):
  File "step19/recursive.py", line 9, in <module>
    dd=ddd+1
TypeError: must be str, not int
```

```
Traceback (most recent call last):
  File "step19/recursive.py", line 9, in <module>
    dd=ddd/0
ZeroDivisionError: division by zero
```

```
Traceback (most recent call last):
  File "step19/recursive.py", line 10, in <module>
    list1[i]=n+1
IndexError: list assignment index out of range
```

02



异常

➤ 生活中的异常



➤ 程序中的异常



- Bug：程序中隐藏着的一些未被发现的缺陷或问题。

1947年的一天，赫柏对Harvard Mark II设置好17000个继电器进行编程后，他的工作却毁于一只飞进电脑造成短路的飞蛾。在报告中，赫柏用胶条贴上飞蛾，并把“bug”来表示“一个在电脑程序里的错误”

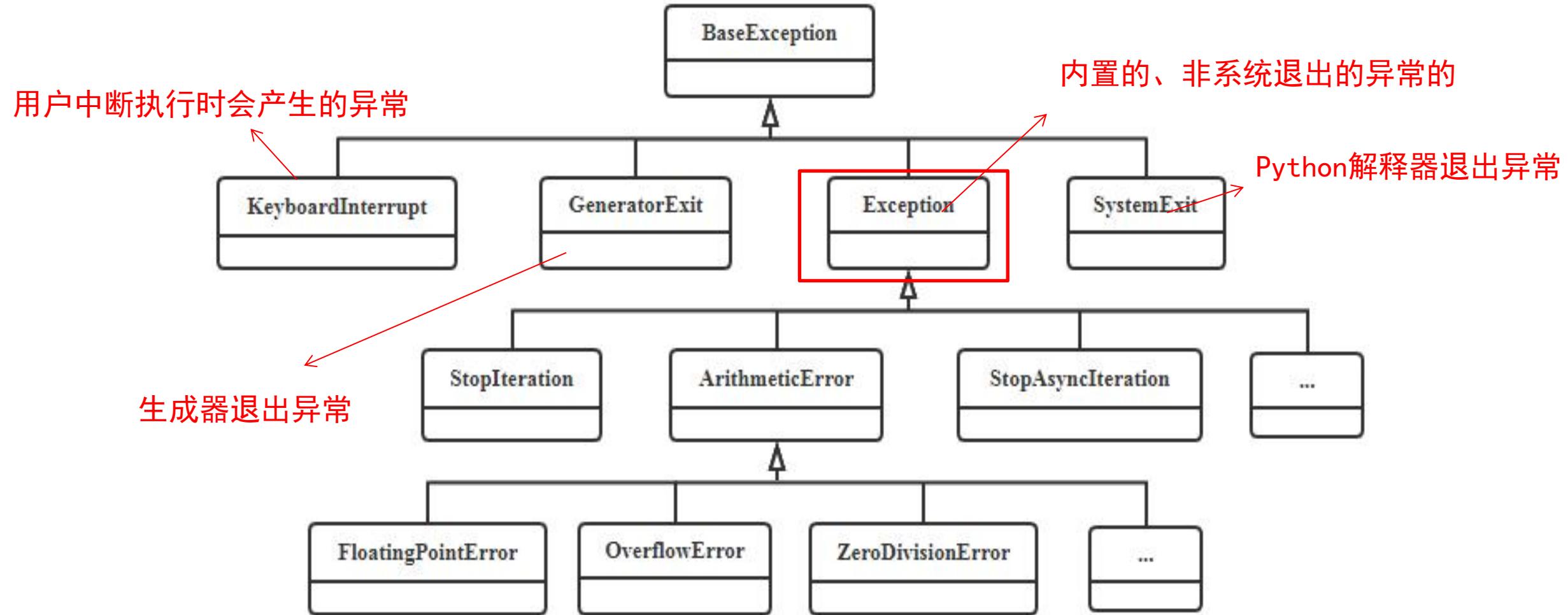


➤ 异常的概念

程序运行出现异常时，若程序中没有设置异常处理功能，解释器会采用系统的默认方式处理异常，即返回异常信息、终止程序。

异常信息中通常包含异常代码所在行号、异常的类型和异常的描述信息。

➤ 异常的类型



➤ 异常的类型

(1) NameError

- NameError是程序中使用了未定义的变量时会引发的异常。
- **例如**，访问一个未声明过的变量ns，代码如下：

—— 实际输出 ——

展示原始输出

```
Traceback (most recent call last):
  File "step19/recursive.py", line 8, in <module>
    ddd=ns
NameError: name 'ns' is not defined
```

➤ 异常的类型

(2) IndexError

- IndexError是程序越界访问时会引发的异常。
- 例如，使用索引访问列表list1，代码如下：

```
Traceback (most recent call last):
  File "step19/recursive.py", line 10, in <module>
    list1[i]=n+1
IndexError: list assignment index out of range
```

➤ 异常的类型

(3) FileNotFoundError

- FileNotFoundError是未找到指定文件或目录时引发的异常。
- **例如**，打开一个本地不存在的文件，代码如下：

```
Traceback (most recent call last):
  File "step19/recursive.py", line 8, in <module>
    f=open("ai.txt",'r')
FileNotFoundError: [Errno 2] No such file or directory: 'ai.txt'
```

➤ 异常的捕获

Python既可以直接通过try-except语句实现简单的异常捕获与处理的功能，也可以将try-except语句与else或finally子句组合实现更强大的异常捕获与处理的功能。

```
try:
```

可能出错的代码

```
# 将捕获到的异常对象赋error
```

```
except [异常类型 [as error]]:
```

捕获异常后的处理代码

语法格式

```
list1 = []
while True:
    try:
        item= input()
        list1.append(item)
    except:
        break
```

try-except语句可以捕获与处理程序的单个、多个或全部异常

```
num_one = int(input("请输入被除数: "))

num_two = int(input("请输入除数: "))

try:
    print("结果为", num_one / num_two)
except ZeroDivisionError: → 单个异常类型
    print("出错了")
```

捕获单个异常

try-except语句可以捕获与处理程序的单个、多个或全部异常

```
num_one = int(input("请输入被除数: "))

num_two = int(input("请输入除数: "))

try:
    print("结果为", num_one / num_two)
except ZeroDivisionError as error:
    print("出错了, 原因: ", error)
```

说明异常原因

try-except语句可以捕获与处理程序的单个、多个或全部异常

```
num_one = int(input("请输入被除数: "))
num_two = int(input("请输入除数: "))
try:
    print("结果为", num_one / num_two)
except (ZeroDivisionError, ValueError) as error:
    print("出错了, 原因: ", error)
```

捕获多个异常

【描述代码执行过程、输出结果】

```
try:  
    print("Step1")  
    a=3/0  
    print("Step2")  
except BaseException as error:  
    print("Step3")  
    print("出错了，原因：", error)  
print("Step4")
```

a=3/10

➤ 异常的else

else子句可以与try-except语句组合成try-except-else结构，若try监控的代码**没有异常**，程序会执行else子句后的代码。

try:

 可能出错的代码

except [异常类型 [as error]]:

 捕获异常后的处理代码

else:

 未捕获异常后的处理代码

语法格式

将捕获到的异常对象赋值error

➤ 异常的else

else子句可以与try-except语句组合成try-except-else结构，若try监控的代码没有异常，程序会执行else子句后的代码。

```
first_num = int(input("请输入被除数: "))
second_num = int(input("请输入除数: "))
try:
    res = first_num/second_num
except ZeroDivisionError as error:
    print('异常原因: ',error)
else:
    print(res)
```

else子句示例

➤ finally

finally子句可以和try-except一起使用，语法格式如下：

try:

 可能出错的代码

except [异常类型 [as error]]:

 捕获异常后的处理代码

finally:

 一定执行的代码

语法格式

将捕获到的异常对象赋值error

➤ finally

- 无论try子句监控的代码是否产生异常，finally子句都会被执行
- finally子句多用于预设资源的清理操作，如关闭文件、关闭网络连接

```
try:  
    file = open('./file.txt', mode='r', encoding='utf-8')  
    print(file.read())  
except FileNotFoundError as error:  
    print(error)  
finally:  
    file.close()  
    print('文件已关闭')
```

finally子句示例

➤ Python程序中的异常不仅可以自动触发异常，而且还可以由开发人员使用**raise**和**assert**语句主动抛出异常。

使用**raise**语句可以显式地抛出异常，**raise**语句的语法格式如下：

raise 异常类	# 格式1：使用异常类名引发指定的异常	raise 语法格式
raise 异常类对象	# 格式2：使用异常类的对象引发指定的异常	
raise	# 格式3：使用刚出现过的异常重新引发异常	

raise **IndexError** 创建异常类对象

示例：raise 异常类

```
IndexError  
<ipython-input-1-55a00e7db5b5> in <module>  
----> 1 raise IndexError
```

```
IndexError:
```

➤ raise

```
raise IndexError(索引下标超出范围)
```

指定异常的具体信息

示例：raise 异常对象

```
IndexError  
<ipython-input-3-3f8088ae2f45> in <module>  
----> 1 raise IndexError('索引下标超出范围')
```

```
IndexError: 索引下标超出范围
```

```
try:  
    raise IndexError  
except:  
    raise
```

示例：重新引发异常

➤ assert

assert语句又称为断言语句，其语法格式如下所示：

assert 表达式[, 异常信息]

assert语法格式

num_one = int(input("请输入被除数："))

示例

num_two = int(input("请输入除数："))

assert num_two != 0, '除数不能为0' #

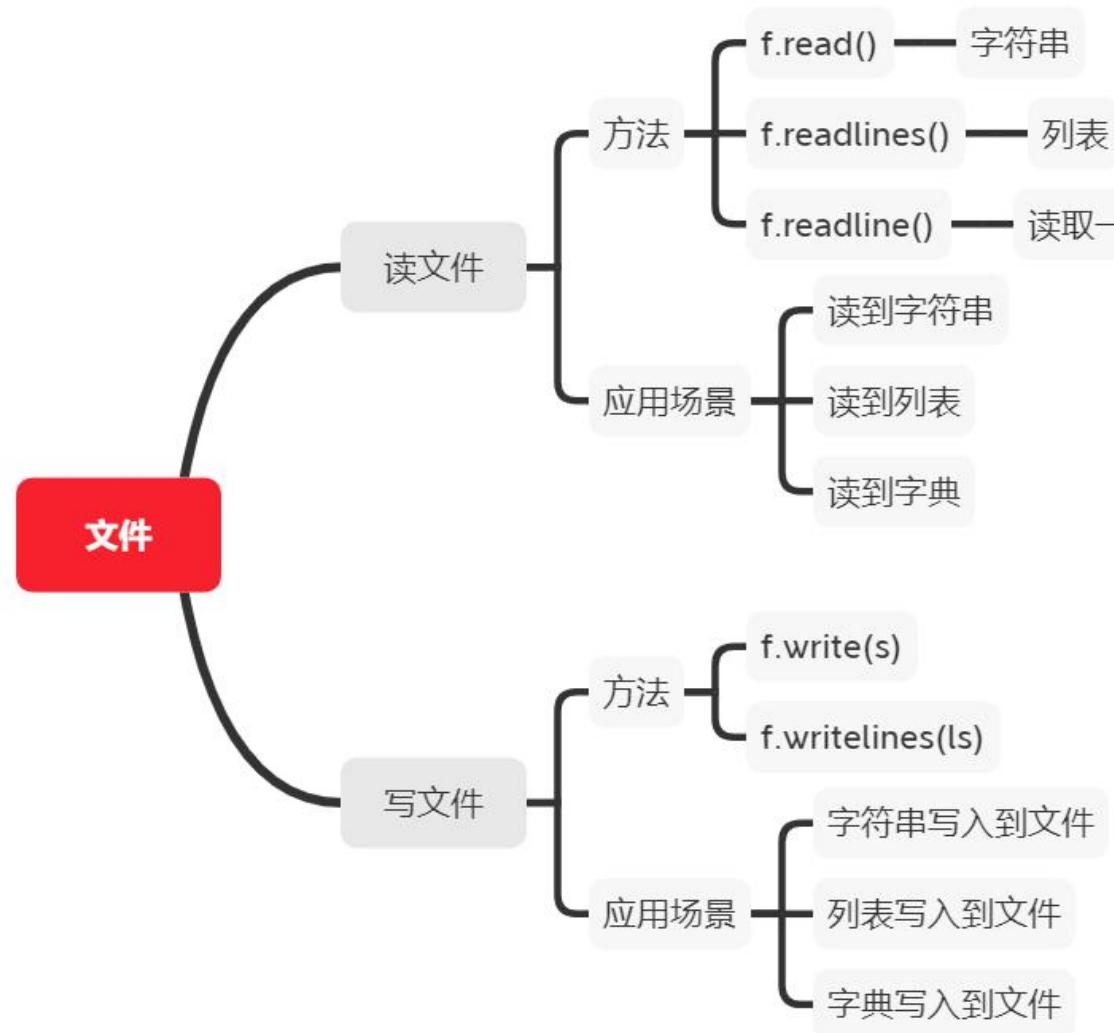
result = num_one / num_two

print(num_one, '/', num_two, '=', result)

请输入被除数：4
请输入除数：0

```
AssertionError                                         Traceback (most recent call last)
<ipython-input-4-e51416c95150> in <module>
      1 num_one = int(input("请输入被除数："))
      2 num_two = int(input("请输入除数："))
----> 3 assert num_two != 0, '除数不能为0' # assert语句判定num_two不等于0
      4 result = num_one / num_two
      5 print(num_one, '/', num_two, '=', result)

AssertionError: 除数不能为0
```



人生启示：

- ✓ 如**文件操作**般守规尽责：读写有度、珍视本源，每一次严谨的行动都是对数据（人生）的守护
- ✓ 似**异常处理**般未雨绸缪：正视问题、主动应对，提前设防方能行稳致远

下课并不代表思考的终止
期待我们下次的思想碰撞

