
# 人工智能程序设计

计算机学院

李晶晶

華南師範大學
South China Normal University

# 问题

- 统计小说walden中各单词出现的频次，并按频次由高到低排序，将统计结果写入文件中。

# 问题分析

- 输入：小说.txt文件
- 处理：用一个函数统计小说中单词出现的频次，并按频次由高到低排序。
- 输出：将统计结果写入文件
- 设计算法
  - 设计函数实现单词频次统计
  - 读文件
  - 写文件

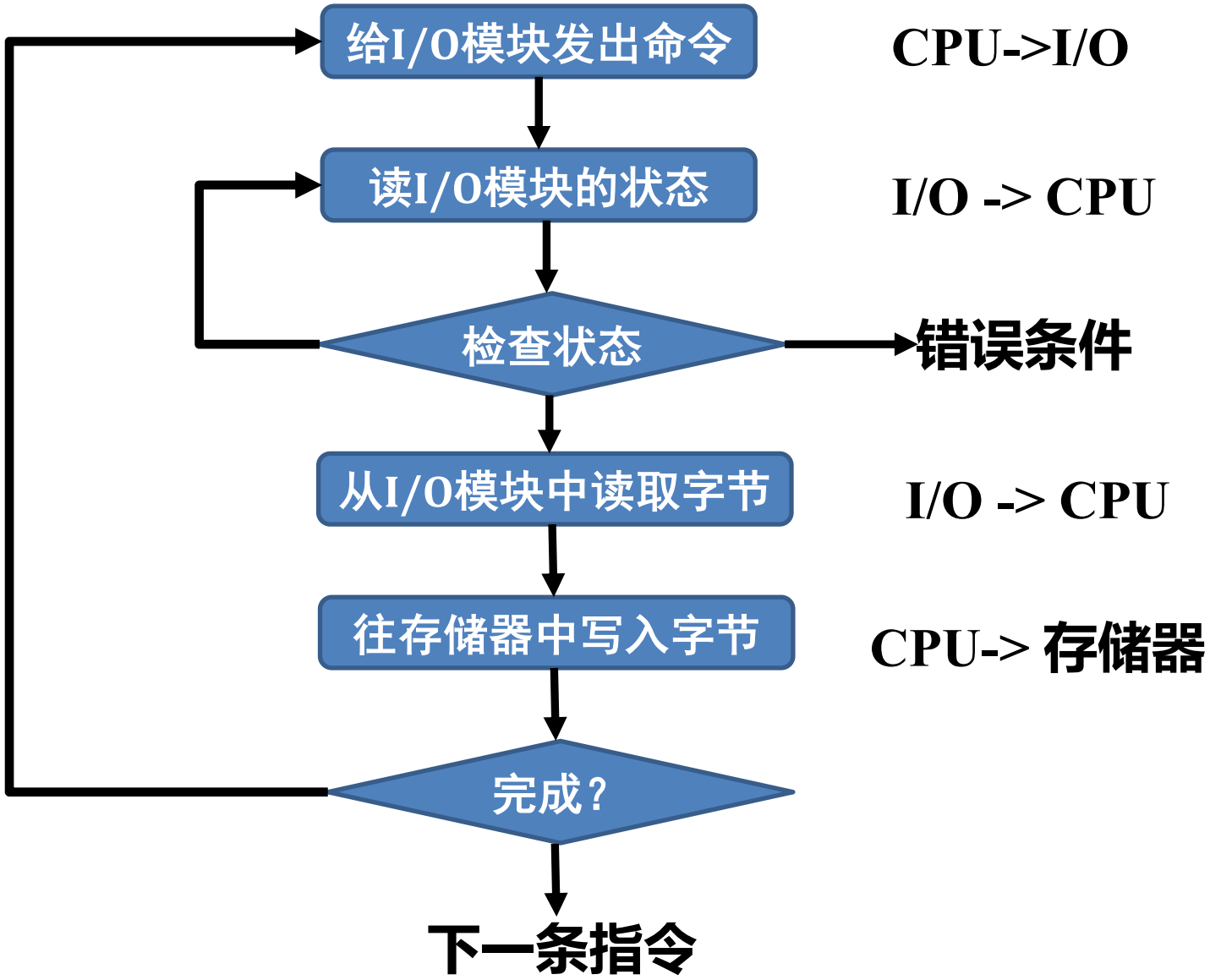

# 目录

- 文件概述
- txt文件
- csv文件
- 错误和异常处理

# 文件概述

- 文件指记录在存储介质上的一组相关信息的集合。
- 文件类型
  - 文件扩展名

# 文件


给I/O模块发出命令
CPU->I/O
读I/O模块的状态
I/O -> CPU
检查状态
错误条件
从I/O模块中读取字节
I/O -> CPU
往存储器中写入字节
CPU-> 存储器
完成？
下一条指令

# 文件


- Python将文本文件视作一个字符序列，而将二进制文件（图像、视频等）视作一个字节序列。
- 对于每一个打开的文件，Python都创建了一个文件对象。

# txt文件

- 以读文件的方式打开一个文件对象，可以使用内置函数open函数传入文件名称与标识符。
- 标识符可指定文件打开模式为读取模式（r）、写入模式（w）、附加模式（a）或读取和写入文件的模式（r+）

# txt文件

- 打开文件

f = open(‘e_1.txt’, ‘r‘)

要读取的文件不存在，或者当前工作路径找不到要读取的文件。

```
Traceback (most recent call last):
  File "test.py", line 1, in <module>
    with open('e_1.txt', 'r') as f:
FileNotFoundError: [Errno 2] No such file or directory: 'e_1.txt'
```

# txt文件

- 打开文件

```
#例6.1:打开文件
f = open('digit_1.txt', 'r')
txt = f.read()
print(txt)
f.close()
```

# txt文件

- 在文件读取过程中，一旦程序抛出IOError错误，后面的close函数将不会被调用。
- 在程序运行过程中，无论是否出错，都要确保能正常关闭文件。

# txt文件

- try...finally...结构

```
#例6.2:try…finally结构
try:
    f = open('digit_1.txt', 'r')
    print(f.read())
finally:
    if f:
        f.close()
```

# txt文件

- with语句读取文件

```
#例6.3:with语句
with open('digit_1.txt', 'r') as f:
    print(f.read())
```

# txt文件

- 设置工作路径
  - 需要打开不在程序文件所属目录下的文件，程序里就需要提供文件所在路径。
  - 相对文件路径

```
#例6.4:相对路径
with open(r‘text_file\e_2.txt', 'r')
as f:
    print(f.read())
```

# txt文件

- 设置工作路径
  - 需要打开不在程序文件所属目录下的文件，程序里就需要提供文件所在路径。
  - 绝对文件路径

```
#例6.5:绝对路径
with open(r‘\Users\Desktop\text_f\e_3.txt', 'r') as f:
    print(f.read())
```

# txt文件

- 创建含有文件数据的列表
  - 读取文件时，检查每一行，查找特定信息，修改文件中文本，可以对文件对象用for循环。
  - 如果需要消除换行符，使用rstrip函数删除string字符串末尾的指定字符（默认为空格）

```
#例6.6:for循环来读取
file_name = 'digit_1.txt'
with open(file_name,'r') as f:
    for line_t in f:
        print(line_t)
```

# txt文件

- 创建含有文件数据的列表
  - 如果需要消除换行符，使用rstrip函数删除string字符串末尾的指定字符（默认为空格）
  - lstrip函数：删除字符前面的指定字符
  - strip函数：删除字符串首尾两端的指定字符

# txt文件

- 创建含有文件数据的列表
  - Read函数可以读取整个文件的内容，读取的内容将存储到一个字符串的变量中。

```
#例6.7:read函数
file_name = 'digit_1.txt'
with open(file_name,'r') as f:
    txts = f.read()
print(type(txts))
```

# txt文件

```
#例6.8:readlines函数
file_name = 'digit_1.txt'
with open(file_name,'r') as f:
    txts = f.readlines()
print(type(txts))
print(txts)
```

# txt文件

- 创建含有文件数据的列表
  - readlines函数可以将读取的文件存储到一个列表里面。
  - 该函数可以实现按行读取整个文件内容，然后将读取的内容存储到一个列表里面。

```
#例6.8:readlines函数
file_name = 'digit_1.txt'
with open(file_name,'r') as f:
    txts = f.readlines()
print(type(txts))
print(txts)
```

# txt文件

- 创建含有文件数据的列表
  - readlines函数可以将读取的文件存储到一个列表里面。
  - 正常打印

```
#例6.9:readlines函数正常打印
file_name = 'digit_1.txt'
with open(file_name,'r') as f:
    txts = f.readlines()
for txt in txts:
    print(txt.strip())
```

# txt文件

```
#例6.10:readline函数
file_name = 'digit_1.txt'
with open(file_name,'r') as f:
    txt = f.readline()
print(type(txt))
print(txt)
```

# txt文件

- 创建含有文件数据的列表
  - readline函数可以实现每次读取文件的一行，将读取到的一行存储到一个字符串变量中，返回字符串类型。

```
#例6.10:readline函数
file_name = 'digit_1.txt'
with open(file_name,'r') as f:
    txt = f.readline()
print(type(txt))
print(txt)
```

# txt文件

- 写入.txt文件
  - 如果要写入的文件不存在，open函数将自动创建文件
  - 如果文件已经存在，以写入模式写入文件时就会先清空该文件。

```
#例6.11:写入txt文件1
file_name = 'words.txt'
f = open(file_name,'w')
f.write('Hello, world!')
f.close()
```

# txt文件

- 写入.txt文件
  - 如果需要将数值型数据写入文本文件，必须先用str函数将数值型数据转换为字符串格式。

```
#例6.12:写入txt文件2
file_name = 'data.txt'
f = open(file_name,'w')
data = list(range(1,11))
f.write(str(data))
f.close()
```

# txt文件

- 写入.txt文件
  - 需要注意的是，当写入多行数据的时候，write函数不会自动添加换行符号，此时会出现几行数据挤在一起的情况。

```
#例6.13:写入多行数据1
file_name = 'words.txt'
f = open(file_name,'w')
f.write("Hello, world!")
f.write("I love Python!")
f.close()
```

# txt文件

- 写入.txt文件
  - 需要注意的是，当写入多行数据的时候，write函数不会自动添加换行符号，此时会出现几行数据挤在一起的情况。
  - 将行与行数据进行区分，需要在write语句内添加换行符号（\n）

```
#例6.14:写入多行数据2
file_name = 'words.txt'
f = open(file_name,'w')
f.write("Hello, world!\n")
f.write("I love Python!\n")
f.close()
```

# txt文件

- 使用with语句写入.txt文件
  - 只有调用close函数，操作系统才保证把没有写入的数据全部写入磁盘。忘记调用close函数的后果可能是数据只写了一部分到磁盘，剩下的丢失了。
  - 所以用with语句更为保险。

```
#例6.15:with语句写入
file_name = 'words.txt'
with open(file_name,'w') as f:
    f.write("Hello, world!\n")
    f.write("I love Python!\n")
```

# txt文件

- 使用with语句写入.txt文件
  - 要写入特定编码的文本文件，需要给open函数传入encoding参数，将字符串自动转换成制定编码。
  - open函数默认enconding参数为UTF-8
  - 要读取非UTF-8编码的文本文件，需要给open函数传入encoding参数。

```
#例6.16:特定编码写入文本文件
file_name = 'words.txt'
f = open(file_name, 'w', encoding='gbk')
f.write('hello world')
```

# txt文件

- 对文件添加内容
  - 给文件添加内容但不覆盖文件原内容，就以附加模式（a）打开文件。此时写入的内容会附加到文件末尾，而不会覆盖原内容。

```
#例6.17:对文件添加内容
file_name = 'words.txt'
with open(file_name, 'a') as f:
    f.write('what\'s your favorite language?\n')
    f.write('My favorite language is Python too.\n')
```

# txt文件

- writelines函数可以把列表中的字符串写入文件
- 注意是连续写入文件，没有换行。

```
#例6.18：wirtelines函数
L = ['abe', 'def']
f = open('words.txt', 'w')
f.writelines(L)
f.close()
```

# txt文件

- print函数不仅能够将信息输出到屏幕上，还可以通过传参的方式让print函数将信息输出到文件中
- print的完整格式：

  print(objects,sep,end,file,flush)，其中objects就是我们平时所需要输出的那些内容，后面4个为关键字参数

# txt文件

- print的完整格式：

  print(objects,sep,end,file,flush)

- sep关键字参数：在输出的字符串之间插入指定字符串，默认是空格，例如

```
#<程序：sep参数举例>
print("a","b","c")              #输出：a b c
print("a","b","c",sep="**")     #输出：a**b**c
```

# txt文件

- print的完整格式：

  print(objects,sep,end,file,flush)

  - end关键字参数：在print输出语句的结尾加上指定字符串，默认是换行(\n)，而如果不想换行则end=''
  - 注意：若不换行，则print语句不会马上打印出当前需要打印的信息，直到需要换行打印的时候才会将这一行信息全部打印出来

# txt文件

- print的完整格式：

  print(objects,sep,end,file,flush)

  - file参数用于将文本输入到某些对象中，可以是文件（注意文件的路径要写对），也可以是数据流等等，默认是输出到屏幕（即sys.stdout）。

```
#例6.19：print到文件中
f = open("words.txt",'r+')
print("a",file=f)   #将“a”输出到文件中
f.close()
```

# txt文件

- print的完整格式：

  print(objects,sep,end,file,flush)

  - flush参数的值只能是True或False，默认为False，表示是否立刻将输出语句输入到参数file指向的对象中 。（False表示在执行f.close()之前内容不会马上被写入文件中）

```
#例6.20：print到文件中
f = open("abc.txt",'w')
print("a",file=f)  #将“a”输出到文件中
```

```
print(“a”, file=f, flush=True)
```

# 上机实践

- 下载瓦尔登（Walden.txt ）文件，将文档保存到和程序设置相同的文件夹中。
- 1）用Python读取文件中的try...finally, with两种方式读取Walden.txt中的数据成列表。
- 2）在桌面新建一个文件夹test，然后将一段瓦尔登湖的文章内容写入test文件夹中的walden_sub.txt中。
- 3）在walde_sub.txt中当前文字后加入当前的年月日的字符串。

# CSV文件

- 逗号分隔值（Comma-Separated Values, CSV）
- CSV文件由任意数目的记录组成，记录间以某种换行符分隔
- 每条记录由字段组成，字段间的分隔符是其他字符或字符串，最常见的分隔符是逗号或制表符。
- 用命令import csv可直接调用csv模块进行CSV文件的读写。

# CSV文件

- 读取CSV文件之前需要用open函数打开文件路径。
- 读取CSV文件的方法有两种
  - csv.reader函数，接受一个可迭代的对象，能返回一个生成器，从其中解析出CSV的内容。

```
#例6.21：CSV文件读取1
import csv
file_name = 'iris.csv'
with open(file_name, 'r') as f:
    reader = csv.reader(f)
    iris = [iris_item for iris_item in reader]
print(iris)
```

# CSV文件

- 读取CSV文件的方法有两种
  - csv.DictReader函数，接受一个可迭代的对象，能返回一个生成器，但是返回的每一个单元格都放在一个字典的值内，而字典的键则是这个单元格的标题（即列头）

```
#例6.22：CSV文件读取2
import csv
file_name = 'iris.csv'
with open(file_name, 'r') as f:
    reader = csv.DictReader(f)
    iris = [iris_item for iris_item in reader]
print(iris)
```

# CSV文件

- 读取CSV文件的方法有两种
  - 如果用csv.DictReader函数读取CSV文件的某一列，则可以用列的标题来查询

```
#例6.23：CSV文件读取某一列
import csv
file_name = 'iris.csv'
with open(file_name, 'r') as f:
    reader = csv.DictReader(f)
    column = [iris_item['sepal length (cm)'] for iris_item in reader]
print(column)
```

# CSV文件

- 写入.CSV文件
  - 对于列表形式的数据，除了cvs.writer函数外，还需要用到writerow函数将数据逐行写入CSV文件。

```
#例6.24：CSV文件写入
import csv
headers = ['Number', 'Name', 'Sex', 'Class', 'Chinese', "Math", 'English']
rows = [('101511', 'Jonh', 'M', 'Class 1', '72', '85', '82'),
        ('101513', 'May', 'F', 'Class 1', '75', '82', '51')]
with open('test1.csv', 'w', newline='') as f:  #打开文件
        f_csv = csv.writer(f)       #创建csv.writer对象
        f_csv.writerow(headers)    #写入1行（标题）
        f_csv.writerows(rows)     #写入多行（数据）
```

# CSV文件

- 写入.CSV文件
  - 字典形式的数据，csv模块提供了csv.DictWriter函数，除了提供open 函数的参数外，还需要输入字典所有键的数据，然后通过writeheader函数在文件内添加标题，标题内容与键一致，最后使用wirterows函数将字典内容写入文件。

# CVS文件

```
#例6.25：CSV文件写入2
import csv
headers = ['Number', 'Name', 'Chinese', "Math", 'English']
rows = [{'Number': '101511', 'Name': 'May', 'Chinese': '72',
'Math': '85', 'English': '82'},
        {'Number': '101513', 'Name': 'John', 'Chinese': '75', 'Math':
'82', 'English': '51'}]
with open('test2.csv', 'w', newline='') as f:  # 打开文件
        f_csv = csv.DictWriter(f, headers)  # 创建csv.DictWriter对象
        f_csv.writeheader()  # 写入标题
        f_csv.writerows(rows)  # 写入多行（数据）
```

# 上机实践

- 1）把csv文件的几个操作的例子实践一遍。
- 2）读取iris.csv文件存储为字典形式的数据后，计算每个属性的均值，并赋值给相应字典的相应键，处理完以后的数据写入新建文件my_iris.csv, 确保数据与属性保持一致。

# 程序的错误


- 语法错误
- 运行时错误
- 逻辑错误

# 异常处理

```
#<程序：程序出现异常示例1>
f = open("file.txt",'r') #文件名为file1.txt,我们却写成file.txt
#输出如下错误：
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'F:/file.txt'
```

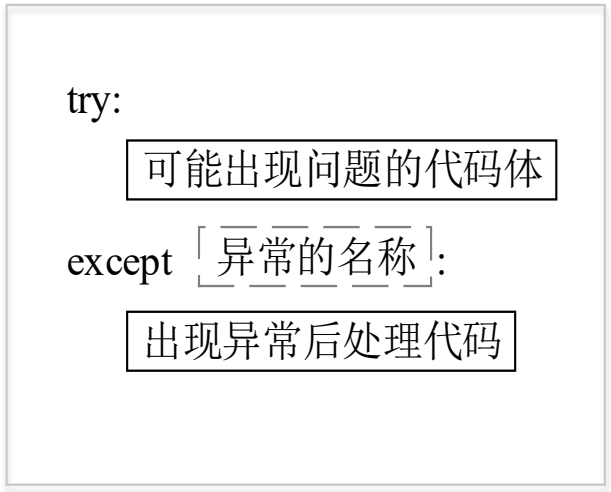
# 异常处理

```
#<程序：程序出现异常示例2>
>>> print(a) #变量a未定义
#屏幕输出如下错误：
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print(a)
NameError: name 'a' is not defined'
```

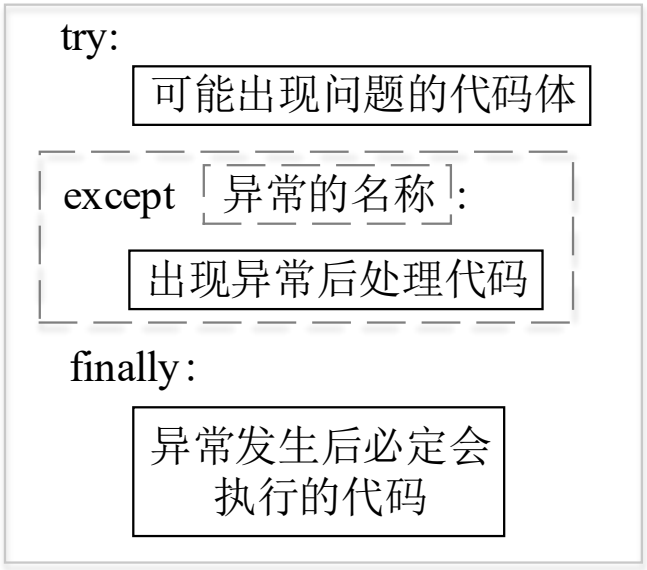
# 异常处理

- 用try语句处理异常
  - 异常捕获可以使用try语句来实现，任何出现在try语句范围内的异常都会被及时捕获到
  - try语句的两种常见形式： try-except、 try-finally

# 异常处理




**try-except模型**




**try-finally模型**

# 异常处理


- try-except模型执行方式如下：
  - 尝试执行try语句内可能出现问题的代码，
  - 如果发现确实出现了我们所写的异常，则执行except部分的处理代码，然后正常执行后面的代码；
  - 否则直接执行try语句中的代码段，然后正常执行后面的代码。

# 异常处理

```
#<例7.5：try-except示例1>
try:
    #文件名为file1.txt,我们却写成file.txt
    f = open("F:/file.txt",'r')
except FileNotFoundError:
    print('文件找不到！！') #最终会输出文件找不到！！
```

```
#<例7.6：try-except示例2>
try:
    #文件存在
    f = open("file1.txt",'r')
    print(a)
except FileNotFoundError:
    print("文件找不到！！")
except NameError:
    print("变量未定义！！")
```

# 异常处理


- 若无法确定要对哪一类异常做处理，也可以这样做

```
...
except:
    print(“出错了！”)
...
```

# 异常处理

- 编写程序，输入一个字符串作为表达式，例如字符串“2+3”，用eval求该字符串的值。假如字符串有除以0，则使用try-except处理异常。

```
#<例7.7：处理除数有0的异常>
try:
    s = input("请输入一个数学表达式：")
    print(eval(s))
except ZeroDivisionError:
    print("除数不可以为0！")
```

# 异常处理

- try-finally模型执行方式如下：
  - 尝试执行try语句内可能出现问题的代码，如果发现确实出现了我们所写的异常，则执行except部分的处理代码，然后必须执行finally部分的代码，再去执行后面其它的代码；
  - 如果没有发现异常，则直接执行try语句中的代码段，跳过except部分，但仍旧要执行finally部分的代码，再去执行后面其它的代码

# 异常处理

```
#<例7.8：try-finally示例1>
try:
    f = open("result.txt", 'r') #文件存在
    print(a)
except NameError:
    print("变量未定义！！")
finally:
    f.close()
```

# 异常处理

- 对try-finally模型来说，except语句部分是可有可无的，如果不写except语句部分，则如果发生异常，会输出Traceback信息，然后执行finally部分的语句，所以我们还可以写出如下程序

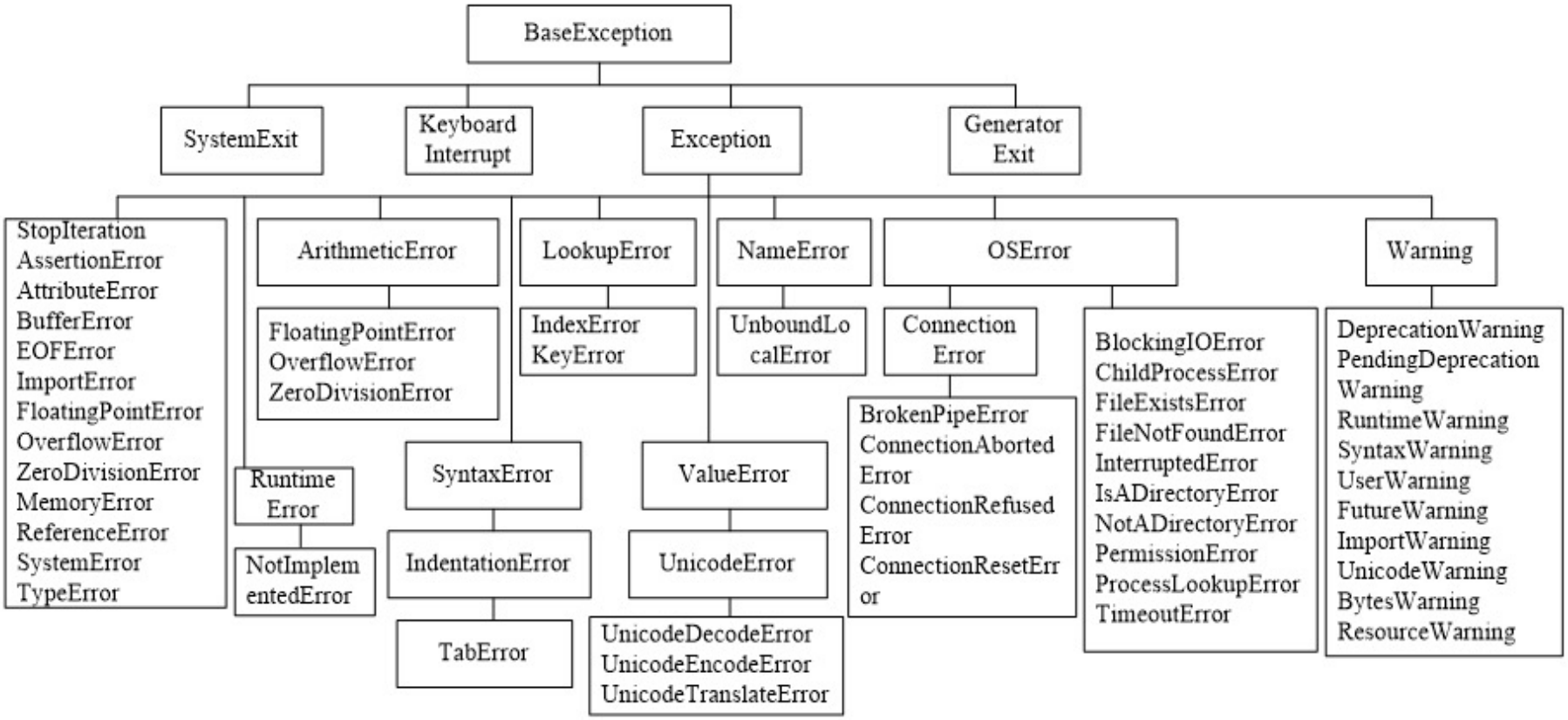
```
#<例7.9：try-finally示例2>
try:
    f = open("result.txt",'r')
    print(a)
finally:
    f.close()
```

# 异常处理

- 常见的异常描述如下表所示

| 异常 | 描述 |
| --- | --- |
| NameError | 尝试访问一个不存在的变量 |
| ZeroDivisionError | 除数为 0 |
| SyntaxError | 语法错误 |
| IndexError | 索引超出序列范围 |
| KeyError | 请求一个不存在的字典关键字 |
| OSError | 操作系统产生的异常，就像打开一个不存在的文件会引发 FileNotFoundError，它就是 OSError 的子类 |
| AttributeError | 尝试访问未知的对象属性 |
| TypeError | 不同类型间的无效操作，比如 1+’1’ |

# 内置异常类


BaseException
SystemExit
Keyboard Interrupt
Exception
Generator Exit
StopIteration
AssertionError
AttributeError
BufferError
EOFError
ImportError
FloatingPointError
OverflowError
ZeroDivisionError
MemoryError
ReferenceError
SystemError
TypeError
ArithmeticError
FloatingPointError
OverflowError
ZeroDivisionError
LookupError
IndexError
KeyError
NameError
UnboundLocalError
OSError
Connection Error
BrokenPipeError
ConnectionAborted Error
ConnectionRefused Error
ConnectionResetError
BlockingIOError
ChildProcessError
FileExistsError
FileNotFoundError
InterruptedError
IsADirectoryError
NotADirectoryError
PermissionError
ProcessLookupError
TimeoutError
Warning
DeprecationWarning
PendingDeprecation Warning
RuntimeWarning
SyntaxWarning
UserWarning
FutureWarning
ImportWarning
UnicodeWarning
BytesWarning
ResourceWarning
Runtime Error
NotImplemented Error
SyntaxError
IndentationError
TabError
ValueError
UnicodeError
UnicodeDecodeError
UnicodeEncodeError
UnicodeTranslateError

# 捕获异常的顺序

- except块可以捕获并处理特定的异常类型（此类型称为“异常筛选器”），具有不同异常筛选器的多个except块可以串联在一起

```
#例7.10:异常类位置顺序示例
a = (44, 78, 90, -80, 55)
total = 0
try:
    for i in a:
        if i < 0: raise
ValueError(str(i)+"为负数")
        total += i
    print('合计=', total)
except Exception:
    print('发生异常')
except ValueError:
    print('数值不能为负')
```

# 自定义异常类

- 自定义异常类一般继承于Exception或其子类。自定义异常类的命名规则一般以Error或Exception为后缀

# 自定义异常类

```
#例7.11: 自定义异常
class NumberError(Exception):  #自定义异常类，继承于Exception
    def __init__(self,data):
        Exception.__init__(self, data)
        self.data = data
    def __str__(self):        #重载__str__方法
        return self.data + ': 非法数值(< 0)'
def total(data):
    total = 0
    for i in data:
        if i < 0: raise NumberError(str(i))
        total += i
    return total
#测试代码
data1 = (44, 78, 90, 80, 55)
print('总计=', total(data1))
data2 = (44, 78, 90, -80, 55)
print('总计=', total(data2))
```

# 程序演示

- 演示代码
  - 输入：小说.txt文件
  - 处理：用一个函数统计小说中单词出现的频次，并按频次由高到低排序。
  - 输出：将统计结果写入文件
  - 设计算法
    - ✓设计函数实现单词频次统计
    - ✓读文件
    - ✓写文件

# 小结


- 文件概述
- txt文件
- csv文件
- 异常与错误