

CSC 171 - Introduction to Computer Programming

Lecture #8 – Files and Exceptions

What is a file?

- A file is a collection of data that is stored on secondary storage like a disk or a thumb drive.
- Accessing a file means establishing a connection between the file and the program and moving data between the two.

Two types of files

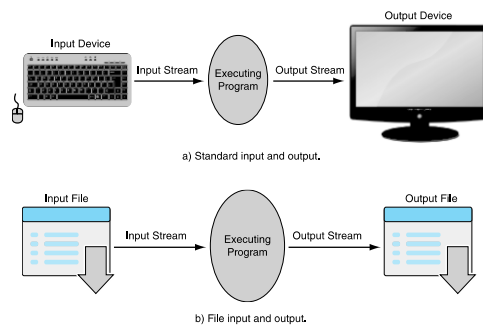
Files come in two general types:

- Text files
 - Files where control characters such as "`\n`" are translated.
 - These are generally human readable
- Binary files
 - All the information is taken directly without translation.
 - Not readable and contains non-readable info.

File Objects or Streams

- When opening a file, you create a file object or file stream that is a connection between the file information on disk and the program.
- The stream contains a buffer of the information from the file, and provides the information to the program.

Input-Output Streams



Buffering

- Reading from a disk is very slow. Thus the computer will read a lot of data from a file in the hopes that, if you need the data in the future, it will be buffered in the file object.
- This means that the file object contains a copy of information from the file called a cache (pronounced "cash")

Making a file object

```
my_file = open("my_file.txt", "r")
```

- `my_file` is the file object.
 - It contains the buffer of information.
 - The open function creates the connection between the disk file and the file object.
 - The first quoted string is the file name on disk, the second is the mode to open it (here, "r" means to read)

Where Is The Disk File?

- When opened, the name of the file can come in one of two forms:
 - `"file.txt"` assumes the file name is file.txt and it is located in the current program directory
 - `"c:\bill\file.txt"` is the fully qualified file name and includes the directory information

File Modes

Mode	How Opened	File Exists	File Does Not Exist
'r'	read-only	Opens that file	Error
'w'	write-only	Clears the file contents	Creates and opens a new file
'a'	write-only	File contents left intact and new data appended at file's end	Creates and opens a new file
'r+'	read and write	Reads and overwrites from the file's beginning	Error
'w+'	read and write	Clears the file contents	Creates and opens a new file
'a+'	read and write	File contents left intact and read and write at file's end	Creates and opens a new file

Careful With Write Modes

- Be careful if you open a file with the `'w'` mode. It sets an existing file's contents to be empty, destroying any existing data.
- The `'a'` mode is nicer, allowing you to write to the end of an existing file without changing the existing contents.

Text Files Use Strings

- If you are interacting with text files (which is all we will do in this course), remember that *everything is a string*.
 - Everything read is a string
 - If you write to a file, you can only write a string

Writing To A File

- Once you have created a file object, opened for reading, you can use the print command
- You add `file=file` to the print command:

```
# Open file for writing:
#     Creates file if it does not exist
#     Overwrites file if it does exist
temp_file = open("temp.txt", "w")
print("First line", file=temp_file)
print("Second line", file = temp_file)
temp_file.close()
```

Close

- When the program is finished with a file, we `close` the file:
 - Closing flushes the buffer contents from the computer to the file
 - Closing tears down the connection to the file
 - `close` is a method of a file obj
 - `file_obj.close()`
 - All files should be closed!

Example: Reversing the text in a File Line by Line

- The program will read text from a file and reverse each line, one at a time:
- Algorithm:
 1. Open the input and output files
 2. Read in each line, reverse it and print it
 3. Close the files

Refining the Reversal Algorithm

1. Open the input and output files
2. Read in each line, reverse it and print it
3. Close the files

-
- 1.1 Open the input file
 - 1.2 Open the output file

Refining the Reversal Algorithm

- 1.1 Open the input file
- 1.2 Open the output file

2. Read in each line, reverse it and print it
3. Close the files

-
- 2 For each line of the file:
 - 2.1 Reverse the characters
 - 2.2 Print the line and the reversed line

Refining the Reversal Algorithm

- 1.1 Open the input file
- 1.2 Open the output file
- 2 For each line of the file:
 - 2.1 Reverse the characters
 - 2.2 Print the line and the reversed line
3. Close the files

- 2.1.1 Start with an empty reversed line
- 2.1.2 Strip the leading and trailing spaces from the line
- 2.1.3 Concatenate each character in the original line to the front of the reversed line

Refining the Reversal Algorithm

- 1.1 Open the input file
- 1.2 Open the output file
- 2 For each line of the file:
 - 2.1.1 Start with an empty reversed line
 - 2.1.2 Strip the leading and trailing spaces from the line
 - 2.1.3 For each character in the line:
 - 2.1.3.1 Concatenate the next character at the beginning of the reversed line
 - 2.2 Print the line and the reversed line
3. Close the files

```
input_file = open("input.txt", "r")
output_file = open("output.txt", "w")
```

Refining the Reversal Algorithm

```
input_file = open("input.txt", "r")
output_file = open("output.txt", "w")
```

2 For each line of the file:

- 2.1.1 Start with an empty reversed line
- 2.1.2 Strip the leading and trailing spaces from the line
- 2.1.3 For each character in the line:
 - 2.1.3.1 Concatenate the next character at the beginning of the reversed line
- 2.2 Print the line and the reversed line
- 3. Close the files

```
for line_str in input_file:
```

Refining the Reversal Algorithm

```
input_file = open("input.txt", "r")
output_file = open("output.txt", "w")
for line_str in input_file:
```

2.1.1 Start with an empty reversed line

- 2.1.2 Strip the leading and trailing spaces from the line
- 2.1.3 For each character in the line:
 - 2.1.3.1 Concatenate the next character at the beginning of the reversed line
- 2.2 Print the line and the reversed line
- 3. Close the files

```
new_str = ''
```

Refining the Reversal Algorithm

```
input_file = open("input.txt", "r")
output_file = open("output.txt", "w")
for line_str in input_file:
    new_str = ''
```

2.1.2 Strip the leading and trailing spaces from the line

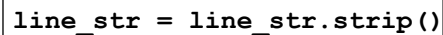
2.1.3 For each character in the line:

2.1.3.1 Concatenate the next character at the beginning of the reversed line

2.2 Print the line and the reversed line

3. Close the files

```
line_str = line_str.strip()
```



Refining the Reversal Algorithm

```
input_file = open("input.txt", "r")
output_file = open("output.txt", "w")
for line_str in input_file:
    new_str = ''
    line_str = line_str.strip()
```

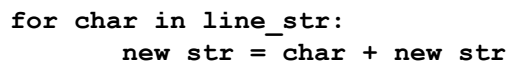
2.1.3 For each character in the line:

2.1.3.1 Concatenate the next character at the beginning of the reversed line

2.2 Print the line and the reversed line

3. Close the files

```
for char in line_str:
    new_str = char + new_str
```



Refining the Reversal Algorithm

```
input_file = open("input.txt", "r")
output_file = open("output.txt", "w")
for line_str in input_file:
    new_str = ''
    line_str = line_str.strip()
    for char in line_str:
        new_str = char + new_str
```

2.2 Print the line and the reversed line

3. Close the files

```
print(new_str, file=output_file)

print("The line is : \"%12s\"\n reversed is:" \"%5s\"\n" \
      % (line_str, new_str))
```

Refining the Reversal Algorithm

```
input_file = open("input.txt", "r")
output_file = open("output.txt", "w")
for line_str in input_file:
    new_str = ''
    line_str = line_str.strip()
    for char in line_str:
        new_str = char + new_str
    print(new_str, file=output_file)
    print("The line is : \"%12s\"\n reversed is:" \"%5s\"\n" \
          % (line_str, new_str))
```

3. Close the files

```
input_file.close()
output_file.close()
```

ReverseFileLines.py

```
# Reversing the contents of a text file
# one line at a time

# Open the input and output files
input_file = open("input.txt", "r")
output_file = open("output.txt", "w")

# For each line in the file
# Start with a blank line
# Strip trailing and leading spaces
for line_str in input_file:
    new_str = ''
    line_str = line_str.strip()

    # For each character in the line
    # Place it at the beginning of the
    # line being created
    for char in line_str:
        new_str = char + new_str

    # Print the line in and file
    # and on the screen
    print(new_str, file=output_file)

    print("The line is : \"%12s\"\n reversed is : \"%5s\"\n" \
          % (line_str, new_str))

# Close the files
input_file.close()
output_file.close()
```

Word Puzzle

- The following listings show how one might solve the following puzzle:
- look through a file of words, one word per line,
- identify any word that has all the vowels in order, with only one example of each vowel.
- For example, "facetious"

Algorithm for Word Puzzle

The initial algorithm is:

1. Open the file
2. For each word in the file:
 - 2.1 Clean it up
 - 2.2 Make sure it's long enough
 - 2.3 Check to see if vowels appear in order

Refining the Word Puzzle Algorithm

1. Open the file
2. For each word in the file:

2.1 Clean it up

2.2 Make sure it's long enough

2.3 Check to see if vowels appear in order

```
word = clean_word(word)
```

Refining the Word Puzzle Algorithm

1. Open the file
2. For each word in the file:

```
word = clean_word(word)
```

2.2 Make sure it's long enough

2.3 Check to see if vowels appear in order


```
2.2 If the word is less than or equal to 6 character  
2.2.1 skip to the next word
```

Refining the Word Puzzle Algorithm

1. Open the file
2. For each word in the file:
`word = clean_word(word)`
 - 2.2 If the word is less than or equal to 6 character
 - 2.2.1 skip to the next word

2.3 Check to see if vowels appear in order

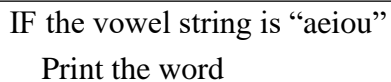
```
vowel_str = get_vowels_in_word(word)
IF the vowel string is "aeiou"
    Print the word
```



Refining the Word Puzzle Algorithm

1. Open the file
 2. For each word in the file:
`word = clean_word(word)`
 - 2.1 If the word is less than or equal to 6 character
 - 2.1.1 skip to the next word
- ```
vowel_str = get_vowels_in_word(word)
```

```
IF the vowel string is "aeiou"
 Print the word
```



```
if vowel_str == 'aeiou':
 print(word)
```



## Refining the Word Puzzle Algorithm

1. Open the file
2. For each word in the file:

```
word = clean_word(word)
```

2.1 If the word is less than or equal to 6 character

2.1.1 skip to the next word

```
vowel_str = get_vowels_in_word(word)
if vowel_str == 'aeiou':
 print(word)
```

```
if len(word) <= 6 :
 continue
```

## Refining the Word Puzzle Algorithm

1. Open the file

2. For each word in the file:

```
word = clean_word(word)
if len(word) <= 6 :
 continue
vowel_str = get_vowels_in_word(word)
if vowel_str == 'aeiou':
 print(word)
```

```
for word in data_file :
```

## Refining the Word Puzzle Algorithm

1. Open the file

```
for word in data_file :
 word = clean_word(word)
 if len(word) <= 6 :
 continue
 vowel_str = get_vowels_in_word(word)
 if vowel_str == 'aeiou':
 print(word)
```

```
data_file = open("dictionary.txt", "r")
```

## Refining the Word Puzzle Algorithm

```
data_file = open("dictionary.txt", "r")
for word in data_file :
 word = clean_word(word)
 if len(word) <= 6 :
 continue
 vowel_str = get_vowels_in_word(word)
 if vowel_str == 'aeiou':
 print(word)
```

## The `clean_word` Function

```
def clean_word(word):
 # Return word in lower case stripped of
 # whitespace."
 return word.strip().lower()
```

## The `get_vowels_in_word` Function

```
def get_vowels_in_word(word):
 # Return vowels in string word—include
 # repeats."
 vowel_str = "aeiou"
 vowels_in_word = ""
 for char in word:
 if char in vowel_str:
 vowels_in_word += char
 return vowels_in_word
```

## WordPuzzle.py

```
Find a word with a single example of the vowels
a, e, i, o, u in that order
```

```
data_file = open("dictionary.txt", "r")
```

```
def clean_word(word):
 """Return word in lower case stripped of
 whitespace."""
 return word.strip().lower()
```

```
def get_vowels_in_word(word):
 """Return vowels in string word-
 -include repeats."""
 vowel_str = "aeiou"
 vowels_in_word = ""
 for char in word:
 if char in vowel_str:
 vowels_in_word += char
 return vowels_in_word
```

```

main program
print("Find words containing vowels 'aeiou' in", \
 " that order:")

for each word in the file
for word in data_file :
 # clean the word
 word = clean_word(word)

 # if the word is too small, skip it
 if len(word) <= 6 :
 continue
 # get vowels in word
 vowel_str = get_vowels_in_word(word)
 # check all vowels in order
 if vowel_str == 'aeiou':
 print(word)|

```

## How To Deal With Problems

- Most modern languages provide methods to deal with ‘exceptional’ situations
- Gives the programmer the option to keep the user from having the program stop without warning
- Again, this is not about fundamental Computer Science, but about doing a better job as a programmer

## What Counts As Exceptional?

- Errors
  - Indexing past the end of a list
  - Trying to open a nonexistent file
  - Fetching a nonexistent key from a dictionary
  - etc.

## What Counts As Exceptional?

- Events
  - Search algorithm doesn't find a value (not really an error)
  - Mail message arrives
  - Queue event occurs

## Other Exceptions

- Ending conditions
  - File should be closed at the end of processing
  - List should be sorted after being filled
- Weird stuff
  - For rare events
  - Keep from clogging your code with lots of if statements.

## Error Names

- Errors have specific names, and Python shows them to us all the time.

```
>>> input_file = open("no_such_file.txt", 'r')
Traceback (most recent call last):
 File "<pyshell#0>", line 1, in <module>
 input_file = open("no_such_file.txt", 'r')
IOError: [Errno 2] No such file or directory: 'no_such_file.txt'
>>> my_int = int('a string')
Traceback (most recent call last):
 File "<pyshell#1>", line 1, in <module>
 my_int = int('a string')
ValueError: invalid literal for int() with base 10: 'a string'
>>>
```

- You can recreate an error to find the correct name. Spelling counts!

# A Kind of Non-local Control

Basic idea:

- Keep watch on a particular section of code
- If we get an exception, raise/throw that exception (let it be known)
- Look for a catcher that can handle that kind of exception
- If found, handle it, otherwise let python handle it (which usually halts the program)

## Doing Better with Input

- In general, we have assumed that the input we receive is correct (from a file, from the user).
- This is almost never true. There is always the chance that the input could be wrong
- Our programs should be able to handle this.



## Worse Yet, Input is Evil

- "Writing Secure Code", by Howard and LeBlanc
  - *“All input is evil until proven otherwise”*
- Most security holes in programs are based on assumptions programmers make about input
- Secure programs protect themselves from evil input

## General Form

```
try:
 suite
except a_particular_error:
 suite
```

## **try** Suite

- The **try** suite contains code that we want to monitor for errors during its execution.
- If an error occurs anywhere in that **try** suite, Python looks for a handler that can deal with the error.
- If no special handler exists, Python handles it, meaning the program halts and with an error message as we have seen so many times.

## **except** Suite

- An **except** suite (perhaps multiple **except** suites) is associated with a `try` suite.
- Each exception names a type of exception it is monitoring for.
- If the error that occurs in the **try** suite matches the type of exception, then that **except** suite is activated.

# try/except Group

- If no exception in the **try** suite, skip all the **try/except** to the next line of code
- If an error occurs in a **try** suite, look for the right exception
- If found, run that **except** suite and then skip past the **try/except** group to the next line of code
- If no exception handling found, give the error to Python

## ExceptDemo.py

```
read a particular line from a file. User
provides both the line
number and the file name

file_str = input("Open what file:")
find_line_str = input("Which line (integer):")

try:
 # potential user error
 input_file = open(file_str)

 # potential user error
 find_line_int = int(find_line_str)
 line_count_int = 1
```

```
for line_str in input_file:
 if line_count_int == find_line_int:
 print("Line {} of file {} is {}".\
 format(find_line_int, file_str,\
 line_str))
 break
 line_count_int += 1
else:
 # get here if line sought doesn't
 # exist
 print("Line {} of file {} not found".\
 format(find_line_int, file_str))
input_file.close()
```

```
except FileNotFoundError:
 print("The file", file_str , "doesn't exist.")

except ValueError:
 print("Line", find_line_str, \
 " Isn't a legal line number.")

print("End of the program")
```