CS 130R: Programming in Python

#11: File manipulation
Reading: Chapter 7
Outline

• File manipulation
  – Text ASCII files
  – Binary files - pickle

• Exceptions
Files store useful data in an meaningful easy way to find them (organized).
File Manipulation

Files – purpose is to store large amount of data in an organized way for efficient retrieval. Make data persistent.

• Types of files:
  – Text - ASCII
  – Binary
  – Images, Movies, Databases
  – Html, Zip, Etc...

Text – ASCII file - contains only ASCII characters.

• Why text files?
  – Text files are cross-platform (same on Windows, Linux, Mac)
  – Text files are easy to use
File Manipulation

• In python, a file is just a string (often very large!) stored on your file system, that you can read or write, gradually or all together.

• Suppose we have a text file like this:
Reading from Text Files

• Opening a file: `open(filename, [access mode])`

```python
text_file = open("read_it.txt", "r")
```

When you open the files, you can specify with parameters for the access mode:

`- "r"` for reading, which tells Python that I want to open the file for reading, if
  the file doesn’t exist, Python will complain with an error.
`- "w"` for writing, if the file exists, its contents are overwritten.
  If the file doesn’t exist, it’s created.
`- "a"` for appending, if file exists, new data is appended to it.
  If the file doesn’t exist, it’s created.

When you open a file the reading cursor is positioned on the first character of
the file so you can read data from the beginning of the file.
Reading from Text Files

• Closing a file: `close()
  text_file.close()`

Note: When you are done with a file, it is very important to close the file.
Reading from Text Files

• Reading characters from a file

Try to run the following sample code:

```python
# Code to read characters from a file

with open('read_it.txt', 'r') as text_file:
    print(text_file.read(1))
    print(text_file.read(5))

# Code to read the whole file

with open('read_it.txt', 'r') as text_file:
    whole_thing = text_file.read()
    print(whole_thing)
```
Reading from Text Files

• Reading characters from a file: \texttt{read([size])}

- \texttt{read}: this function reads all characters as a big string, unless you specify the number of characters that you want to read

  e.g. \texttt{text\_file.read()}  \#read all characters as a big string

  or

  \texttt{text\_file.read(1)}  \#read the first character

Note that we have a reading cursor for the current opened file
Reading from Text Files

• Reading characters from a line: `readline([size])`

Example 1:
```python
text_file = open("read_it.txt", "r")
print(text_file.readline(1)) # read the first character of the current line
print(text_file.readline(5)) # read the next five characters of the current line
```
```python
text_file.close()
```

Note: `readline()` reads characters from the current line only, while `read()` reads characters from the entire file

Example 2:
```python
text_file = open("read_it.txt", "r")
print(text_file.readline()) # read 1st line
print(text_file.readline()) # read 2nd line
print(text_file.readline()) # read 3rd line
```
Reading from Text Files

• Reading all lines into a list: \texttt{readlines()}

```python

text_file = open("read_it.txt", "r")
lines = text_file.readlines()
print(lines)
```

Running result:

```python
['Line 1
', 'This is line 2
', 'That makes this line 3
']
```
Reading from Text Files

- Looping through a file
  ```python
text_file = open("read_it.txt", "r")
for line in text_file:
    print(line)
text_file.close()
```

Or
  ```python
text_file = open("read_it.txt", "r")
listLines = text_file.readlines()
for line in listLines:
    print(line)
```

Running result:
```
Line 1
This is line 2
That makes this line 3
```
Writing to text files

• Writing strings to a file: `write(output)`

```python
text_file = open("write_it.txt", "w")

    text_file.write("Line 1\n")
    text_file.write("This is line 2\n")
    text_file.write("That makes this line 3\n")

Or

text_file = open("write_it.txt", "w")

    text_file.write("Line 1\nThis is line 2\nThat makes this line 3\n")
```

Note: The write method doesn’t automatically put the newline character at the end of the string it writes. These need to be put manually.
Writing to text files

• Writing a list of strings to a file: `writelines(output_list)`

```python
text_file = open("/Users/haoranli/Desktop/Python_class/write_it.txt", "w")

lines = ["Line 1\n", "This is line 2\n", "That makes this line 3\n"]

lines = writelines(lines)

text_file.close()

# in the file write_it.txt

Line 1
This is line 2
That makes this line 3
```

The `writelines` method doesn’t automatically put the newline character at the end of the string it writes. These need to be put manually.
## Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>file_object = open(filename, [access mode])</code></td>
<td>Open a file, if it doesn’t exist, create a new file</td>
</tr>
<tr>
<td><code>file_object.close()</code></td>
<td>Close the file</td>
</tr>
<tr>
<td><code>file_object.read([size])</code></td>
<td>Reads <code>size</code> characters from a file and returns them as a string. If <code>size</code> is not specified, returns all of the characters from the current position to the end of file.</td>
</tr>
<tr>
<td><code>file_object.readline([size])</code></td>
<td>Reads <code>size</code> characters from the current line in a file and returns them as a string. If <code>size</code> is not specified, returns all of the characters from the current position to the end of the line.</td>
</tr>
<tr>
<td><code>file_object.readlines()</code></td>
<td>Reads all of the lines in a file and returns them as elements in a list</td>
</tr>
<tr>
<td><code>file_object.write(output)</code></td>
<td>Writes the string <code>output</code> to a file</td>
</tr>
<tr>
<td><code>file_object.writelines(output_list)</code></td>
<td>Writes the strings in the list <code>output_list</code> to a file</td>
</tr>
</tbody>
</table>
Next lecture

• Binary files
• Exceptions ......