# Files, Format method, and Other Useful Stuff

Professor Hugh C. Lauer
CS-1004 — Introduction to Programming for Non-Majors

(Slides include materials from *Python Programming: An Introduction to Computer Science*, 2nd edition, by John Zelle and copyright notes by Prof. George Heineman of Worcester Polytechnic Institute)

## Today

- Introduction to files in Python
- String methods

## Definition — File

- **A (potentially) large amount of information that lives a (potentially) very long time**
- **May be (much) larger than the amount of RAM in your computer**
- **(Usually) expected to outlive the running of your program**
- **(May be) expected to outlive the computer itself!**
- **Stored on**
  - Hard drive
  - Flash drive
  - Spread out across multiple disks
  - Somewhere in the "cloud"
  - On some other medium
  - …

## **Files** (continued)

- **(Usually) stored as a sequence of *bytes***
  - *Byte:* an 8-bit character
  - The standard unit of storage since 1964
  - Other data types built up from sequences of bytes
- **Organization of data within file defined by application**
  - Text
  - Numerical data
  - Big databases
  - Program code
  - …
  - Directory (a.k.a. folder) — special kind of file containing list of names and locations of other files
    - Owned and maintained by operating system

## Using Files

- **Must be *Opened* before use**
  - Tells OS to make file ready for access
- **Must be *Closed* when finished**
  - Tells OS to "put the file away"
  - Make it safe for long term storage
- **Note: Most operating systems automatically close files that are still open when program exits**

**Don't depend on this!**

- **Stale data may live in volatile memory for long time**
  - Where it can become corrupted …
  - … or forgotten …
  - … or lost
  - … before OS gets around to writing to disk!

**Remember to close your files before exiting your program!**

**Three other modes:–**

- 't' – text mode (default)
- 'b' – binary
- '+' – update in place

## Open

- **Gets a file ready for use**
  - OS sets up internal tables
  - May fetch copy off remote disk
  - Validates protection,
  - Etc.

```
f = open(filename, mode)
```

- **Built-in function**
- **Filename**
  - String of text
  - Name of the file (as seen in directory), with extension
  - Possibly including directory “path”
- **Mode**
  - 'r' – read (default)
  - 'w' – write (truncate to zero length)
  - 'a' – append
  - … (other modes — see *Python* documentation)

## Close

```
f.close()
```

- **File method**
- **Closes the file**
  - Clears internal buffers
  - I.e., puts it away safely
- **Don’t forget to do this in your program!**
  - Penalty for forgetting!

## Reading from text files

`f = open(filename, mode)`

`f.read()`

- **Reads *entire* remaining contents of file into one (potentially humungous) string**
  - Line endings represented by '\n' characters
  - "Remaining" means from where we left off reading most recently to end of file

`f.read(n)`

- **Reads *n* characters from the file.**

`f.readline()`

- **Reads one line**
  - Including '\n' character (a.k.a. *newline* character)
  - Returns line as a string

`f.readlines()`

- **Reads all remaining lines**
  - Returns a list of strings
  - Each representing one line — as from **`readline()`**

## Iterating thru a file

- F = open(fileName, ‘r’)

- for line in F:

```
        # line is a string ending with '\n'
        # do something with this line

        print(line)
                                        or
        print(line[:-1])   #without trailing '\n'
```

## Writing to a file

```
f = open(outputfilename, 'w')
f = open(outputfilename, 'a')
```

- 'w' truncates file — i.e., removes existing contents
- 'a' appends to file — i.e., preserves existing contents

```
f.write(string)
```

- **Writes the string at the end of the file**
  - Returns number of bytes written
- ***You* need to supply trailing '\n' character**
  - To denote end of line
- **You *may* write partial lines**
  - i.e., with no trailing '\n'
- **You *may* write multiple lines at one time**

**Until you are more skilled,**
**concentrate on writing one full line at a time!**

## Alternative way to write to file

Must refer to a file object opened for writing!

- **oFile = open('fileName.txt', 'w')**
- **print('string', file=oFile)**
  - Example
  - See p. 156 (bottom)
- **Similar to**
  **oFile.write ('string')**
- **print function by default adds '\n'**
  - **end=** default parameter
  - write method does not
- **print function accepts multiple strings**
  - E.g print(s1, s2, …, nN)
  - Separated by default by spaces
  - **sep=' '** default parameter
  - write method does ???

## What next?

- **Close the file!**

- **Note:– both *Python* and *OS* keep contents of file buffered in memory**
  - I.e., volatile memory!
  - Closing flushes the buffers to disk
  - Where it is stored safe from (most) failures

## Questions?

## Today

- **Introduction to files in Python**

- **String methods**
  - **`string.format()`**

  - **`string.split()`**
  - **`string.join()`**

  - **`string.strip()`**

  - **`string.lstrip()`**
  - **`string.rstrip()`**

  - … (more on p. 140)

## `string.format()`

- **Simple use of `string.format()`**

```
T = "Hello {0} {1}, you may have won ${2}"

T.format('Mr.', 'Smith', 1000) ⇒
 'Hello Mr. Smith, you may have won $1000'
```

- **Definitions:–**
  - *Template string:–* a string with *replacement fields* delimited by braces (i.e., curly brackets)
  - *Replacement field:--*

    **`{<index>:<format-specifier>}`**
  - **`index:`**– position of argument to **`format()`** method
    - *Empty index means "Use the next argument in order"*

- **Meaning:–**
  - **Make and return a copy of *template string* in which each *replacement field* is replaced by the value of the argument numbered by *index* …**
  - **… formatted according to the *format-specifier***

## `string.format()` (continued)

- **The following are equivalent:–**

```
T = "{0}{1}{2}"
T.format(pi, sqrt(2), 0)
```

**and**

```
"{0}{1}{2}".format(pi, sqrt(2), 0)
```

- **Reason:– `format()` method can apply to *any* string, constant or variable!**
  - Second version used heavily in textbook §5.8.2

## Format specifiers

- **An entire sub-language**

- **Examples:–**

**`{0}`**

  - No format for argument *0* specified
  - Use default formatting for that type
  - Take as much space as needed

**`{2:5}`**

  - Format argument *2* to take at least 5 spaces
    - (More if needed)

**`{1:7.5}`**

  - Format argument *1* to take at least 7 spaces with five <u>total</u> <u>digits</u> of precision

**`{1:7.5f}`**

  - Format argument *1* to take at least 7 spaces with five decimal digits <u>*after*</u> decimal point

## Format specifiers (continued)

- **How to line up numbers?**

```
    2 won
    1 wreaths
    5 you
    1 your
    1 you’ve
  113 Distinct words
```

`"{0:>5} {1}"`

  - Argument *0* in five spaces, right justified
  - Argument *1* with default formatting (a string!)

- **Left justifying, centering**
  - `{:<5}.format(…)`
  - `{:^5}.format(…)`

- **Aligning decimal points**
  - `"{:8.4f}".format(…)`

## Questions?

## More string methods

- **See p. 140, Table 5.2**
- **`string.split(chars)`**
  - Split into substrings
  - Any character in **`chars`** delimits a split
    - Defaults to “white space” — i.e., tabs, spaces, newlines, etc.
  - Returns list of strings
- **`string.strip(chars)`**
  - Remove any sequence of characters in **`chars`** from beginning or end of string
  - Returns a new string
- **`string.lstrip(chars)`**
  **`string.rstrip(chars)`**
  - Remove any sequence of characters in **`chars`** from beginning OR end of string
  - Returns a new string

## Questions?

## Useful tidbits

**`import os`**

- **`os.listdir()`**
  - Lists the current directory
- **`os.listdir(path)`**
  - Lists the directory found at **`path`**
- **`os.getcwd()`**
  - Gets the current working directory
- **`os.chdir(path)`**
  - Changes the current working directory to **`path`**
- **`os.mkdir(path)`**
  - Creates a new directory with name **`path`**
  - Absolute or relative to current working directory
- **Lots of other tidbits**

## Useful menu items in *IDLE*

- **Path browser**
  - Shows the various directories that *Python* searches to find modules, etc.
  - Listed in order of search
  - See example
- **Class browser**
  - Shows the classes and functions defined in current module
  - Click to get to definition
- **Open Module …**
  - Tries to find and open the module by searching the *path*
  - Opens *Python* modules but not built-in internal modules

## Questions?