Files and Directories

Opening a file

- To work with a file, you must (1) **open** the file, (2) **use** the file (read, write, or append), and (3) **close** the file
- To open a file, call **open(***path_to_file*, *mode***)**
  - The *path_to_file* is a ***string***; it can be
    - Just the name of a file, if in the same directory as the program
    - An absolute path, starting with **/** or with something like **C:**
    - A relative path, containing **/**s, but not starting with one
- To write to a *text* file, the mode can be one of
  - **'r'** to indicate you just want to read the file
  - **'w'** to indicate you want to write to the file
    - Warning: If the file already exists, this will replace it
  - **'a'** to indicate you want to append (write) to the end of an already existing file
- To use a binary file, use **'rb'**, **'wb'**, or **'ab'**
- The **open** function returns a "file object," or ***stream***, that you use subsequently

Files and streams

- To be very precise:
  - A ***file*** is outside your program, on the file system
  - A ***path*** is a string that specifies the location of a file
  - A ***stream*** is an object in your program that *connects* to a source of data, or a destination for data--such as a file
    - Your keyboard is a source of data (**sys.stdin**), and your screen is a destination for data (**sys.stdout**), so we could treat them as streams
- We often aren't that precise, and it’s common to see a variable named **file** used to hold a stream

Absolute paths

- An ***absolute path*** to a file specifies exactly where the file is on *your* computer
  - On Windows, it starts with drive number, such as **C:**
  - On other systems, it starts with a **/**, indicating the root directory
- If your program uses an absolute path, it almost certainly won't work on any other computer
- **Moral:** Don't use absolute paths! (At least, not without a good reason)
- **Tip:** In a path name, **'.'** means "the current directory," so starting a path with **'./'** means "starting from the current directory"
  - Also, **'..'** means "the directory containing the current directory"

Closing files

- You should always close a file when you are done using it
  - You can't open it again until it has been closed
  - If you were writing to it, an unclosed file may be incomplete
- There are two ways to do this:
  - *stream* **= open(***path***,** *mode***)**
    statements using the stream
    stream.**close()**
  - **with open(***path***,** *mode***) as** *stream***:**
    statements using the stream
- **with** is not a loop or a conditional statement
  - All **with** does is close the file after the statements have been executed

Reading files

- The **open** method (with mode '**r**') returns a ***stream*** that you can use to read lines from that file
  - You can read the file one line at a time:
    line **=** *stream*.**readline()**
  - You can read the file as a single long string (containing **\n** characters):
    lines **=** *stream*.**read()**
  - You can read the file as a list of lines:
    line **=** *stream*.**readlines()**
  - With the above methods, you must remember to **close** the stream
- You can treat the stream as a sequence of lines with a **for** loop:

```
for line in open(path, 'r'):
    # code using the line in the line variable
```

  - The for loop uses the stream, so you don't have to
  - The stream will be closed automatically when you exit the **for** loop

Newlines

- One character that has never been completely standardized is the “newline” character
  - “Classic” Macintosh used to use the “carriage return” character, octal 15 (**\r**)
  - UNIX and Linux use the “line feed” character, octal 12 (**\n**)
  - Macintosh is now based on UNIX, and uses line feed
  - Windows uses carriage return followed by line feed (**\r\n**)
- To cope with this, when Python reads something it recognizes as a newline, it turns it into **\n**
- When Python writes a newline, it writes the appropriate kind of newline for the operating system that it is running on
- **Bottom line:** You *usually* don’t have to worry about this...
  - ...but if you are reading or writing *binary* files, you do!
- **Tip:** Use *line*.**rstrip()** to remove the **\n** at the end of lines you read in

Writing files

- The **open** method (with mode '**w**') returns a stream that you can use to write lines to that file
- You can write a **string** *s* to a file with *stream*.**write(***s***)**
- Most simple values can be converted to a string with the **str** function: *stream*.**write(str(***value***))**

Asking about streams

- If *s* is a stream, then:
  - **s.name** is the name of the file associated with the stream
  - **s.mode** is the mode in which the file was opened
  - *s.closed* is *True* if the file has been closed
  - **s.readable()** is **True** if the file can be read
  - **s.writable()** is **True** if the file can be written to

Where am I?

- Since relative file paths are relative to the ***current directory***, it is useful to know what that directory is
  - **Note:** ***Directory*** and ***folder*** are synonyms
- First, **import os**
  - (**from os import *** does *not* do what is required)
- Then, **os.getcwd()** will return the "current working directory" as a string, for example, **'/Users/dave/Documents'**
- **os.chdir(***path_to_directory***)** will change the current working directory to the given one
  - The *path_to_directory* must specify a directory, not a file

Directories

- **os.listdir(*path*)** returns a list of all the entries (files and directories) in the specified directory
  - Default value of *path* is **'.'**, meaning the current working directory
- **os.mkdir(*path*)** creates a directory with the given *path* name
- **os.mkdirs(*path*)** is like **mkdir**, but also creates all directories along the path if they don't already exist
- **os.rmdir(*path*)** removes an *empty* directory, or raises an **OSError** if the path specifies a non-empty directory
  - **shutil.rmtree(*path*)** can be used to recursively delete a directory and everything in it
- **os.rename(*src*, *dst*)** changes the name of the file or directory from *src* to *dst*
- These are just the methods that I have found most useful; there are many more in **os** and **shutil**

Paths

- Suppose **p = '/Users/dave/TEMP/test.py'**, then:
  - **os.path.split(p)** returns **('/Users/dave/TEMP', 'test.py')**
  - **os.path.dirname(p)** returns **'/Users/dave/TEMP'**
  - **os.path.basename(p)** returns **'test.py'**
  - **os.path.join('/Users/dave/TEMP', 'test.py')** returns **‘/Users/dave/TEMP/test.py'**
  - **os.path.exists(p)** returns **True** if **p** refers to an actual file or directory
  - **os.path.getsize(p)** returns the size of file or directory **p**
  - **os.path.isdir(p)** returns **True** if **p** refers to a directory
  - **os.path.isfile(p)** returns **True** if **p** refers to a plain file

Pickle

- You can save any Python object to a file in binary, and read it in again later
  - ```
    # Save a dictionary into a pickle file.
    import pickle

    favorite_color = { "lion": "yellow", "kitty": "red" }
    pickle.dump( favorite_color, open( "save.p", "wb" ) )
    ```
  - ```
    # Load the dictionary back from the pickle file.
    import pickle

    favorite_color = pickle.load( open( "save.p", "rb" ) )
    # favorite_color is now { "lion": "yellow", "kitty": "red" }
    ```
- This example taken directly from https://wiki.python.org/moin/UsingPickle

The End

- More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity.

--W.A. Wulf