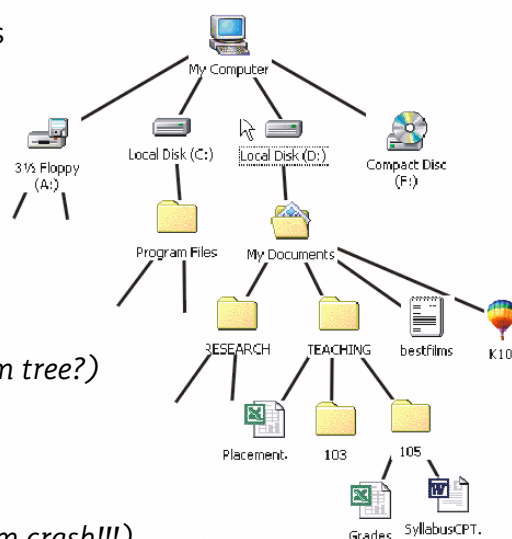


Working with Files and Directories

Introduction

Today, we will look at how Python can read from and write data files. This is a fundamental skill in programming because we use files to store data when our programmes are not running, or to share data between programmes. We will cover:

- File system tree
 - current working directory
(*where am I?*)
 - absolute versus relative paths
(*how do I refer to another location in file system tree?*)
- File properties
 - does it exist?
(*can't open/read it if it does not exist!! program crash!!!*)
 - is it a file or a directory?
(*a directory contains other files/directories while a file contains data.*)
 - is it binary or text file?
 - is it open/closed?
(*can only read/write contents when the file is open*)
- Searching for files by name.
- Copying versus moving files.
- Deleting files and directories.
- Reading and writing files.



In today's session we are going to implement a number of file related tasks in python — these tasks are small but cover typical problems when working with files.



We will need a collection of files that we don't care about to experiment with. Download the zip archive from the website and extract to your coderdojo_tramore directory.



1 File/Directory — Cheat Sheet

The following is a summary of the most important file/directory related commands that we use.

1.1 Important modules

- `os` — used to determine current working directory, check directory/file properties.
- `glob` — used to search for files by filename.
- `shutil` — used to copy or move/rename directories/files.

1.2 Current working directory



- `os.getcwd()` --- **return** the current working directory (as a string)
- `os.chdir(name)` — change the current working directory to name.

1.3 Directory and file functions


Do I exist and what am I?

- `os.path.exists(name)` — returns True if directory/file name exists, false otherwise.
- `os.path.isdir(name)` — returns True if name is a directory, false otherwise.
- `os.path.isfile(name)` — returns True if name is a file, false otherwise.

How big am I?

- `os.path.getsize(name)` — returns the size of file name.  File must exist.
- `os.listdir(name)` — returns a list of files in name.  Must be a directory.
- `os.path.isfile(name)` — returns True if name is a file, false otherwise.

Coping or moving/renameing ...

- `shutil.copy(source, target)` — copies a single file source to target.  File source must exist.
- `shutil.move(source, target)` — move/rename directory/file source to target.

Creating a directory ...

- `os.mkdir(name)` — create directory  Directory must not exist.

Deleting directories/files ... do I need to say that this is dangerous?

- `os.remove(name)` — delete file name  File must exist.
- `os.rmdir(name)` — delete director name  directory must exist and be empty.


Finding files ...

- `glob.glob(pattern)` — find all files that match the given pattern.
- `os.walk(name)` — a directory tree generator. Lists all directories/file in name, then in sub=directories, then sub-sub-directories, etc.



1.4 Reading and writing functions

Before any read/write we need to open the file ...

- `open(name, mode)` — open file name for reading or writing. The string mode can be:
 - `open(name, "r")` — open file name for reading in text mode (default).
 - `open(name, "wt")` — open file name for writing in text mode  existing contents are deleted
 - `open(name, "at")` — open file name for appending in text mode.

Reading from a file ... after opening a file, called name, using `f = open(name)`

- `data = f.read()` — read a single, next line from file.
- `data = f.readlines()` — read all lines from file. returns a list or lines.

Writing to a file ... after opening a file, called name, using

- `f.write(data)` — write string in data to file.
- `f.writelines(data)` — write list of strings in data to file.

After all reads/writes close the file ...

- `close(f)`

1.5 Cleaning/Processing data from files

We often need to tidy up data (string) we get from files. Given a string `s`, typical tasks are.

- `len(s)` — return number of characters in string `s`.
- `s.strip()` — return a copy of the string, `s`, with leading and trailing whitespace removed.
- `s.split(char)` — splits string `s` whenever character `char` is found. Splits at spaces if `char` is not given. Returns a list of strings.

1.6 Extra concepts

- Keyword `with`.

Python programmers can get into trouble if they forget to close an opened file. The `with` keyword was introduced into python to address this. We won't — we should, but we won't — use this today, but we will switch to reading/writing files using `with` soon.

- Module `shelve` allows saving and reading of python data. We will cover this at a later time.
 - Module `shelve` using a separate module called, `pickle` to convert python data into a format that can be saved to files. Many programmers use `pickle` directly, but we will stick to `shelve`.
- Module `csv` supports reading and writing CSV (Comma separated value) files.



2 Programming Tasks

Note: Switch to Python3 mode in mu-editor for these tasks.

We will never get all of these done in two hours — I have include more so that you can practise during the week, if you like.



`print_zen.py`

Create a programme, called `print_zen.py` which prints out the contents of the file `data/The_Zen_of_Python.txt`.



`print_binary.py`

Create a programme, called `print_binary.py` which tries to print out the contents of the file `images/dot.png`. This is a binary¹ file and printing out contents of binary files is usually not a good idea.

In directory `data` I have saved a few limericks, one per file.



`print_limericks.py`

Write a programme called `print_limericks.py` that finds all files in folder `data` with name matching pattern `limerick_*.txt`, and prints out the file name followed by the limerick it contains.



`tidy_limericks.py`

Write a programme called `tidy_limericks.py` that

- Creates sub-directory `limericks` in directory `data`.
- Copies each of the limericks files from directory `data` into sub-directory `limericks`.



`untidy_limericks.py`

Write a programme called `untidy_limericks.py` that effectively undoes the work of our `tidy_limericks.py` code above:

- Removes each of the individual limerick file in sub-directory `limericks`.
- Removes sub-directory `limericks` from directory `data`.



`merge_limericks.py`

Write a programme called `merge_limericks.py` that copies the contents of all of the limericks in files from directory `data` into a single file `data/all_limericks.txt`.

¹Text versus binary files:

- Text files — These have text in them, with letters, numbers, punctuation, and some special characters, like newlines.
- Binary files — These don't have text in them. They might have music, pictures, or some kind of data, but because they don't have text, they don't have lines either, because there are no newlines.



count_words_in_zen.py

Write a programme called `count_words_in_zen.py` that creates a new file called `zen_count.txt` in directory `data` that contains the number of words per each line in the file `data/The_Zen_of_Python.txt`.



longest_line_in_file.py

Create a programme, called `longest_line_in_file.py` which prints out the line with the most characters in the file `data/The_Zen_of_Python.txt`.