PROGRAMMING IN PYTHON I

Unit 05: Files and paths

Michael Widrich
Institute for Machine Learning

FILES

Files

- We already heard that we store information in bits
  - Integer, float, strings, . . .
  - Images, formatted text, audio data, . . .
- We agree on one encoding and decoding scheme for the bits
  - jpeg, mp3, pdf, . . .
- We can then store these bits as file on a storage device
- Typically, we have a file system that organizes our files
  - Keeps track of where which file is stored and how it can be stored/retrieved
  - Our OS usually supports many file systems with differences between OS

File type and file suffix

- Our files have a base filename to identify them
  - E.g.: `myfile`
- To indicate the type (encoding/decoding schema) to our OS, we can use a file suffix
  - Typical format: `filename.filesuffix`
  - E.g.: `my_textfile.txt`, `my_picture.jpg`, ...
  - This file suffix is part of the filename and it's only an indication

  → we could rename `my_picture.jpg` to `my_picture.txt`

FOLDERS AND PATHS

Folders and paths

- Many file systems support directories (a.k.a. folders)
  - Allows for grouping of multiple files into one folder
  - Often hierarchical (folders can contain subfolders, which can contain subsubfolders, etc.)
  - In Linux, folders are also (special) files, containing a list of entries and folder ID
  - In Linux/Unix, the first directory is called root directory
  - Linux uses the / character to separate directories:
    `folder/subfolder/subsubfolder/myfile.txt`

Absolute and relative paths

- The folder we are currently in is called working directory
  - Linux: Command `pwd` prints current working directory
- The location of a file can be specified as absolute or relative path
- Absolute filepaths
  - Includes root directory
  - Independent of current working directory
  - E.g.: `/home/sam/folder/myfile.txt`
- Relative filepaths
  - Start from some given working directory
  - Avoids absolute paths
  - E.g.: `folder/myfile.txt`

FILES IN PYTHON

Files in Python (1)

- In Python, we can open a file using `open()`
- Usage: `filehandle=open(filename: str, mode: str)`
- `filehandle`: object that allows us to interact with file content (it is not the file content itself!)
  - `filehandle.seek()`: Move stream position (starting position for reading and writing) to position in file
  - `filehandle.read()`: Read content of file
  - `filehandle.write('text')`: Write something to file
  - For more functions see code for Unit 05
- `filename`: Path to file (relative or absolute)

Files in Python (2)

- Usage: `filehandle=open(filename: str, mode: str)`
- `mode`: What do we want to do with the file?
  - 'r' Read-only (read from file, fails if file does not exist)
  - 'w' (Over)Write-only (write to file, create new file if it does not exist or delete original file content if file is already existing)
  - 'a' Append-only (write to file, create new file if it does not exist but keep original file bits/append new content to end of file)
  - 'r+' Read and write (read from file and write new content to beginning of file, fails if file does not exist)
  - 'a+' Read and append (read from file and write new content to end of file, create new file if it does not exist)
- Default stream positions:
  - `'r'`, `'w'`, `'r+'`: Beginning of file
  - `'a'`, `'a+'`: End of file

Files in Python (3)

- Usage: `filehandle=open(filename: str, mode: str)`
- `mode`: Also specifies if it's a text file or not
  - Text mode
    - File is interpreted as string
    - Returns string objects when reading from file
    - Expects string objects to write to file
    - Modes: `'r'`, `'w'`, `'a'`, `'r+'`, `'a+'`
  - Binary mode
    - File is not interpreted
    - Returns bytes objects when reading from file
    - Expects bytes objects to write to file
    - Modes: `'rb'`, `'wb'`, `'ab'`, `'rb+'`, `'ab+'`
- More information:

  `https://docs.python.org/3.6/tutorial/inputoutput.html#reading-and-writing-files`

IMPORTANT HINTS

Important hints (1)

- When you open a file, you also have to close it
  - What you write to a file is buffered and not necessarily written to the file until it is closed
  - Your file might be corrupted if you terminate the program before the file is closed
- If your program is aborted (e.g. by user or exceptions), the file might not be closed correctly
- The with block will close the file automatically and should be used where possible

Important hints (2)

- Using relative paths increases portability of your code
  - The path `/home/sam/folder/myfile.txt` might exist on Sam's computer but on Andi's computer it would not work
  - The path `folder/myfile.txt` works on Sam's and Andi's computer as long as they start from the correct working directory (e.g. `/home/sam/` and `/home/andi/otherfolder/`)
- Directory separators might be different between different OS
  - The os module provides functions which allow for OS independent path handling
  - E.g.: `os.path.join('folder', 'myfile.txt')` will create `folder/myfile.txt` or `folder\myfile.txt` depending on the used OS