PROGRAMMING IN PYTHON I

Files

Andreas Schörgenhumer

Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ

Institute for
Machine Learning

Copyright Statement

Contact

Andreas Schörgenhumer

Institute for Machine Learning
Johannes Kepler University
Altenberger Str. 69
A-4040 Linz

E-Mail: schoergenhumer@ml.jku.at

Write mails only for personal questions

Institute ML Homepage

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 a **file** on a storage device
- Typically, we have a **file system** that organizes our files
  - Keeps track of all files and how they can be stored/retrieved
  - Our operating system (OS) usually supports many file systems with differences between OS
  - Normally, you do not have to worry about the file system

File Type and File Suffix/Extension

- 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 (file extension)
  - Typical format: `filename.filesuffix`
  - E.g.: `my_textfile.txt`, `my_picture.jpg`, `...`
  - This file suffix is part of the filename and is only an indication (so the OS chooses an appropriate program to open the file)
  - → We could rename `my_picture.jpg` to `my_picture.txt`
- Make sure that your OS is configured to show the file extensions, so you can see them

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.)
- Hierarchies are OS-dependent:
  - In Unix, the first directory is called **root directory**
  - Unix uses the `/` character to separate directories:
    `/folder/subfolder/subsubfolder/myfile.txt`
  - In Windows, the first directory is your hard drive (or partition) ID, e.g., **C:**
  - Windows uses the `\` character to separate directories:
    `C:\folder\subfolder\subsubfolder\myfile.txt`

Absolute and Relative Paths

- The folder we are currently in is called **working directory**
- The location of a file can be specified as **absolute** or **relative path**
- Absolute file paths:
  - Include the root directory
  - Are independent of the current working directory
  - Example: `/home/sam/folder/myfile.txt`
- Relative file paths:
  - Start from some given working directory
  - Avoid absolute paths
  - Example (assume the current working directory is `/home/sam`): `folder/myfile.txt`

Files in Python (1)

- In Python, we can open a file using the built-in function **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
- `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 is a text file or not
  - Text mode:
    - File is interpreted as string (using a specified character encoding, e.g., UTF-8)
    - 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+"`

Opening and Closing Files)

- 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 flushed and 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
- We could solve this with `try-finally` blocks where we guarantee that the file is closed properly
- Or we use the **with** statement that will close the file automatically. This should be used where possible (shorter and more convenient than `try-finally`)

Example of Reading From a File

```
# Default mode is reading "r"
with open("myfile.txt") as f:
    content_as_string = f.read()

# File is properly closed after the "with", do
   something with "content_as_string"
```

- Note that the `with` statement does not handle any exceptions, e.g., if the specified file is missing
- It only ensures that the file is properly closed, if it was opened before

Portability

- 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 differ between different OS
  - The **os** module provides functions which allow for OS independent path handling
  - Example: `os.path.join("folder", "myfile.txt")` will create `folder/myfile.txt` or `folder\myfile.txt` depending on the used OS

Storing Python Objects in Files

- **pickle** module
  - Allows us to store and load many types of Python objects
  - Stores data in binary files (serialization) and reconstructs them when reading again (deserialization)
  - Not compressed (unless we compress the file using compression modules)
  - Can handle many different Python objects
- **dill** module (use this one instead of **pickle**)
  - Same interface as **pickle**
  - Extends functionality of **pickle**
  - Can store more types of Python objects
  - Often used as `import dill as pickle`