Python

Module 7

Handling Different File Formats with Python

Bits and Bots
study group

Content

This module

Welcome to the last module of the Python track from the Bits and Bots study group! This module focuses on how to handle different file formats with Python. We handpicked four chapters for you, that you can read and work through. It is not necessary to follow them all. Based on the summary per chapter provided in this module, you can choose what you want to work with and find something that suits your day to day practice best. From web scraping to PDF documents, Python has multiple modules and packages to import into your program to read and write different file formats.

For this module, make sure you:

- Recap what you have learned in module 6. Check your own notes, or read the summaries provided at the end of the chapters of Al Sweigart books.
- Read and work through chapter(s) 13, 14, 17 or 18 of *Automate the Boring Stuff with Python*.

Recap of Module 6

In Python, you can store data in **variables** while your program is running. However, if you want to retain the data after your program is finished, you need to save it to a file. Files have two key properties: a *filename* and a *path*, that specifies the location of a file on your computer. The **/ operator** allows you to combine Path objects and strings.

There are two ways to specify a file path: an **absolute path** always begins with the root folder, such as `C:\`. A **relative path** is relative to the program's current working directory, that is recognisable at the "dots" that are used in the path. You can use methods in Python, to check if a given path exists.

You can read files with the `read()` method in Python and write files with the `print()` function. The last one works the same as how you would write content on your screen, for example in the interactive shell or the command-line interface. To write the content to a file you need to open the write mode: `file = open('example.txt', 'w')`. Do note that if you want to add text to the existing contents of a file, you need to open the file in append mode: `file = open('example.txt', 'a')`.

If you want to copy, move, rename, and delete files in your Python programs, you need to import the shell utilities with `import shutil` in your script. It is good to have some caution when you're working with files, especially when you write in files or delete them. Make sure not to overwrite files if you weren't meant to and that you use the send2trash module to be sure your program is not deleting files that it wasn't supposed to delete. In short, verify exactly what you do and practice a lot in a practice folder with duplicate or dummy files that won't be ruined if you make a mistake.

Pick and Mix

For the final module, we want you to pick your own goal. Therefore, there are four options to choose from. The first option focuses scraping web pages, while the other ones each teach you how to work with different file format:

1. Web scraping
2. Working with spreadsheets
3. Working with PDF and Word documents
4. Working with CSV, JSON, and XML documents

If you're not ready to stop developing with games yet, you can also have a look at the pygame module that is introduced in Chapter 17 in *Invent Your Own Computer Games with Python*.

Installing packages when needed
If you are following one, or more chapters, from *Automate the Boring Stuff with Python*, you might stumble upon some packages that are not automatically installed with Python. You have to do this yourself, using the `pip tool` (that has to be installed by you as well). But don't worry! How to do this is also explained in the book, just check out Appendix A.

1. Web Scraping

The term **Web scraping** is used for a program that downloads and processes content from the web. In chapter 13 of *Automate the Boring Stuff with Python* you will learn more about web scraping and make use of five different modules and packages, which make it easy to scrap web pages in Python. The chapter will cover:

- The **`webbrowser` module.** It comes with Python and opens a browser to a specific page. This module lets users cut out the step of opening the browser and directing themselves to a website. The program does this for you.

- The **`requests` module**, which enables you to download files and webpages from the internet, without you having to worry about network errors, connection routing, and data compression. This module doesn't come with Python, so it has to be installed by yourself with the help of provided instructions in the book. With built-in methods, you can check for errors and save downloaded files to your hard drive. With the requests module it is possible to get information from an API. *Note: If you want to learn how to download video files from websites such as YouTube, Facebook, or X (formerly Twitter), you need to use the yt-dlp module instead. This is covered in chapter 24, jump to 'Downloading Videos from Websites'.*

- The **Beautiful Soup (bs4) package** parses HTML to extract the information you want. Even a simple HTML file involves many different tags and attributes, and matters quickly get confusing when it comes to complex websites. With the bs4 `select() method` you can select one or multiple HTML elements and with the `get() method` you can access HTML attribute values from an element. The chapter includes an example of a program that can automatically follow links to scrape large amounts of data from the web, with the help of the bs4 package.

- The **Selenium package** launches and controls a web browser, such as by filling in forms and simulating mouse clicks, just as a human user would. You have to use this package, instead of the requests module, if you want to interact with a web page that depends on the JavaScript code that updates the page.

- The **Playwright package** launches and controls a web browser. It is newer than Selenium and has some additional features. One of the new features is the ability to simulate a browser without actually having the browser window open on your screen. This makes it useful for running automated tests or web scraping jobs in the background.

Start by reading this blog about web scraping. Then read and work through Chapter 13 of Automate the Boring Stuff with Python.

2. Working with Spreadsheets

When working in digital preservation, you might have stumbled upon a spreadsheet or two. And even if you haven't, almost everyone uses spreadsheets to organise information.

The **openpyxl module** allows your Python programs to read and modify Excel spreadsheet files. This module doesn't come with Python, so it has to be installed by yourself with the help of provided instructions in the book. The fourteenth chapter covers two projects that help you understand how to read (analyse and count data and print the results) and how to write (create, delete and edit spreadsheets) Excel documents with a Python program. The chapter also covers formulas and adjusting rows and columns, such as merging and unmerging cells and freezing rows or columns.

For working with Microsoft Excel: Read and work through Chapter 14 of Automate the Boring Stuff with Python.

For working with Google Sheets: Read and work through Chapter 15 of Automate the Boring Stuff with Python.

Both of the chapters cover **tuples**, another built-in data type comparable with lists and dictionaries. It can be helpful to have a look at Python Tuples on W3Schools.

If you're working with Google Sheets, you have to install **EZSheets** with the pip command line tool, as explained in the book. With the help of the fifteenth chapter, you will learn how to create, download, upload, and delete spreadsheets. Just as in Excel, Google Sheets worksheets have columns and rows of cells containing data. You will learn how to read and write data in a similar way. With EZSheets, it is also possible to access Google Form data from the spreadsheet.

3. PDF and Word Documents

If you want your Python programs to read and write PDFs or Word documents, you'll need to do more than simply pass their filenames to open(). This is because these types of documents are much more complex than simple plaintext files. PDF and Word documents are binary files that also contain font, colour, and layout information next to text. The Python packages **PyPDF** and **Python-Docx** make interactions like reading and writing easy.

- **PyPDF** is a Python package for creating and modifying PDF files. You need to install the package with the help of provided instructions in the book. Word of warning: PDF files are not easy to parse into plaintext, so even PyPDF may simply be unable to work with some of your particular files. If it does work on your file, you might be able to extract text or images from a PDF file, or create PDFs from other pages. Furthermore, you'll learn how to rotate and insert blank pages, how to add watermarks and overlays, and how to encrypt and decrypt PDFs.
- Python can create and modify Microsoft Word Documents with the **Python-Docx package**. You need to install this package with the help of provided instructions in the book. The .docx files have many structural elements, which Python-Docx represents using three different data types: document objects (entire document), paragraph objects (a new paragraph begins whenever the user presses `ENTER` or `RETURN` while typing in a Word document) and run objects (contiguous run of text with the same style). This chapter teaches you how to write in Word files with the help of a Python script, such as adding headings, line and page breaks, and pictures.

Read and work through Chapter 17 of Automate the Boring Stuff with Python: PDF and Word Documents.

Chapter 17 briefly mentions **tuples**, another built-in data type comparable with lists and dictionaries. It can be helpful to have a look at Python Tuples on W3Schools.

4. CSV, JSON, and XML Files

The file formats CSV, JSON and XML are common plaintext formats for storing data. These formats were designed for use by computers. Python can work with them much more easily than PDF and Word files for example, because they are meant to display nicely for human readers. CVS, JSON and XML are more easy for programs to *parse* (that is, read and understand) while still being human readable. Python comes with **`csv`**, **`json`** and **`xml`** **modules** to help you work with these formats. **CSV (Comma-Separated Values)**: a simple way to organise data, kind of like a basic spreadsheet. Each row represents a set of information, and each piece of data in the row is separated by a comma. It works well when you have a lot of similar data, like a list of names and email addresses. If you want to know more about CSV files, you can read this blog about it.

- **JSON (JavaScript Object Notation)**: a format that looks like the way data is written in the JavaScript programming language but you don't need to know JavaScript to understand it. JSON is easy to read and write, and it's often used to send data between computers or websites. It was designed to be simpler than older

formats like XML. If you want to know more about JSON, have a look at JavaScript JSON on W3Schools.

- **XML (Extensible Markup Language)**: a more complex format that has been around for a long time. It uses tags (just like HTML does) to describe the data. While XML is powerful and widely used in large systems, it can be harder to work with if you don't need all its extra features. If you want to know more about XML, you can follow the XML Tutorial on W3Schools.

Read and work through Chapter 18 of Automate the Boring Stuff with Python: CSV, JSON, and XML files.

Introduction to File Format Identification

This month our session will focus on **File Format Identification**. We will cover specific tools and the differences between them, such as **DROID** (Digital Record Object Identification) and **Siegfried**.
We will explore running these tools from the command line and integrating them with Python. We will also cover how to analyze file formats manually and how these tools function using information from file format registries (like PRONOM).
Finally, we will look at resources you can use when trying to understand the files you hold, whether on a personal computer or digital files in a collection.

Why is this useful?

Preservation: You cannot preserve a file if you don't know what it is. Identification is the first step in digital archiving.
Security: Malware often hides by masquerading as a harmless file type (e.g., an .exe renamed to .jpg). Identification tools look at the binary signature, not just the file extension.
Troubleshooting: When a file won't open, it is often because the extension doesn't match the actual format.

Pre-requisites

For this session it might be useful if you have access to a few tools.

1. A Hex Editor

You need a tool to view the raw binary data of a file, here are some tools that you can use, and others are available.
Windows: HxD
macOS: Hex Fiend
Linux: GHex or Bless

2. An Identification Tool

These exercises will focus on DROID and Siegfried. Siegfried has a more accessible command line interface whilst DROID can be more approachable with a GUI.

3. Access to sample files

We have placed some sample files in the Bits and Bots repository, GitHub is also an excellent resource for downloading sample files or you may already have some files sets

that you wish to analyze on your computer. Some resources on finding sample files can be found in the PRONOM Starter Guide.

Some exercises to try out

These exercises are based extensively on the PRONOM Starter Guide and the monthly session (which will be made available online shortly after the session).

Pick one, a couple or all of the following exercises depending on your level of confidence!

1. Install one of the following on your computer and run it over a set of files: DROID or Siegfried. There are comprehensive user manuals for DROID online and Siegfried also has some information available online, but it requires a little more looking through GitHub. Take a careful look at the file set and answer the following questions:
    - What methods are the tools using to identify the files? Do they identify by extension, byte matcher/ signature or container
    - Are there any unidentified files to research further, what might the files be?
    - Are there any flags, extension mismatches or anything to be concerned about? For example did a file that is called image.png actually identify as a .jpg

If you are comfortable with the Command Line Interface (CLI), try installing Siegfried using your package manager.

Windows (using Scoop):
code Powershell
downloadcontent_copy
expand_less
  scoop install siegfried

macOS (using Homebrew):
code Bash
downloadcontent_copy
expand_less
  brew install siegfried

Once installed, verify it works by typing sf -v in your terminal.

Stretch exercise: What else can you do with the command line tool and Siegfried or DROID? Can you create any automated workflows for your files?

2. Research a file format

Are there any file formats in your collection that you can't identify, or do you have an area of interest you want to explore further. PRONOM also has a list of file formats that do not yet contain full descriptions or have signatures. If you want to take on a challenge and create a description for a file format in PRONOM, or (hard mode) attempt to create a signature for a file that doesn't have one then the PRONOM Research GitHub repository

has all the tools and lists you would need in the readme. If you have your own files you would like to explore further then have a go following the steps in the Starter Guide.

3. Find a common pattern within the hex of a set of files.

For this exercise we will be looking at these files, Open the files and drag and drop these into your hex editor. Can you find any consistent patterns within the file format? Is this the "File Signature" or "Magic Number"? Can you work out how these files could be identified? For tips on using a hex editor and analysing files you can use the Starter Guide.

Optional Challenge: Python

For those comfortable with Python, try to write a script that mimics what Siegfried does. Create a python script that iterates through the **Sample Dataset**.
For example:

```
downloadcontent_copy
expand_less
   with open('filename', 'rb') as f:
   print(f.read(4).hex())
```

Create a simple dictionary (Key: Value) where the Hex Signature is the key, and the Format Name is the value.
Print out the format of the files based on your dictionary lookup.