Webmanual2PDF

J.C. Martín

October 4, 2023

Contents

1 Introduction

A lot of manuals are on the Internet. Generally, they are a set of web pages inside a web folder with one index web page on that web folder that refers to the manual web pages. This web folder is the root folder of the manual. The URL of the manual refers to the index web page. Sometimes the URL is the path of the index web page file, and sometimes is the path of the root folder of the manual, in which case the index web page of the manual is the default web page file of the root folder. The manual is a set of web pages inside a structure of web folders, all of them inside the root folder. The web manual folder is the same as the root folder of the manual.

The aim of this project is double. First, it is to know the `href` attribute values of the `a` tags of the index web page referenced by the manual URL. These values are web pages, that are separated into two groups the internal ones and the external ones. One of those web pages is *internal* if its web folder is inside the root folder of the manual. The *external* web pages are the non-internal ones. These internal web pages are the content of the manual. The second aim is to download the information on the internal web pages. This information is stored in an output directory. Each web page in the manual is stored in a PDF file. These PDF files are in a directory tree similar to that of the original root folder of the manual.

Note. Remember to check the permissions in the website's `robots.txt` file.

Since some web pages of the manual are default web page files, and we do not know their names, e.g. `index.html`, or `default.html`, this project uses `[default]` for they and `[default].pdf` for their PDF versions.

This project has two modules and two scripts. The modules are `file_and_dir_utils.py` and `web_manual2pdf_mod.py`. The scripts are `rename_dir_files.py` and `web_manual2pdf.py`. The main script is `web_manual2pdf.py`.

2 The script `web_manual2pdf.py`

2.1 How it works

This script is the main script of the project and is an interface for the following functions: `url_href_info` and `generate_pdfs`. Both of these functions are on the project module `web_manual2pdf_mod.py`.

The function `url_href_info` has only the parameter `url`. This function separates the set of all the URLs that are `href` attribute values of `a` tags of the `url` argument into two lists. The first one with all the URLs that are in the root folder of the `url` argument (the internal URLs), and the second one has the remaining URLs (the external ones). The function returns to the terminal these lists.

Remember that the internal web pages are the manual content, so the first list returned to the terminal by `url_href_info` is the list of the manual web pages.

The function `generate_pdfs` has four parameters, they are

- `url`, a string that gives the manual URL
- `output_dir`, a string that gives a valid folder path name
- `pdf_options`, a dictionary that corresponds to the named parameter `options` of the function `pdfkit.from_url`. As parameter of the function `generate_pdfs`, its default value is `DEFAULT_PDF_OPTIONS`, where

  ```
  DEFAULT_PDF_OPTIONS = {
      'page-size': 'Letter',
      'no-outline': None,
  }
  ```

- `log_filename`, a string that is a valid filename, `'web_manual2pdf.log'` is its default value

This function attempts to create PDF versions of the manual web pages, each one of them in the directory of the local folder tree that corresponds to the manual web folder tree. After that, it creates a log file whose name is `log_filename` and is located at `output_dir`. This folder, `output_dir`, is also the base of the local folder tree.

The script needs five variables to work. These are

- `function`, an integer. Its possible values are the constants `INFO` and `PDFS` (`INFO = 0`, and `PDFS = 1`)

- `input_url`, a string. It must be the web manual URL
- `output_dir`, a string. It must be a valid directory name
- `options`, a dictionary. It must be a valid value for the named parameter `options` of the function `pdfkit.from_url`
- `log_filename` , a string. It must be a valid filename

If `function` is `INFO`, the script calls

`wpm.url_herf_info(input_url)`

if `function` is `PDFS`, the script calls

`wpm.generate_pdfs(input_url, output_dir, options, log_filename)`

That means that `input_url` is the `url` parameter of `url_href_info` and `generate_pdfs`; `output_dir` is `output_dir` of `generate_pdfs`; `options` is `pdf_options` of `generate_pdfs`; and `log_filename` is `log_filename` of `generate_pdfs`.

2.2 The script

The script is the following

```
import web_manual2pdf_mod as wmp

"""
This script is an interface for the functions 'url_href_info' and 'generate_pdfs'
from web_manual2pdf.py. The variable 'function' must be the constant 'INFO' (resp.
'PDFS') for use 'url_href_info' (resp. 'generate_pdfs').

DOCSTRING OF URL_HREF_INFO:
This function separates the href attribute values of the 'a' tags in the url
argument into the ones such that the corresponding url start with url_dir,
which is the url path directory of the url argument, and the others that do
not start with url_dir. Then it prints these two lists into the output.

Requirements: get_internal_and_external_hrefs, get_url_dir, and url2log

:param url: A string
:return: None

DOCSTRING OF GENERATE_PDFS:
This function separates the href attribute values of the 'a' tags in the url
argument into the ones such that the corresponding url start with url_dir,
which is the url path directory of the url argument, and the others that do
not start with url_dir. For the first urls, it uses the package pdfkit to
```

```
produce pdf files of these urls. The relative path of these pdf files begins
with 'output_dir' and the relative url with the extension changed to pdf. If
the url refers to a directory then the base name of the file is '[default].pdf'
instead of '.pdf'. In the same folder, '[default].pdf' can have the same content
as 'index.pdf' or 'default.pdf' if any of the latter exist.

This function also produces a log file in the 'output_dir' directory,
'log_filename' is the filename of this log file. This log file contains three
lists. The list of the href values that generated a pdf file, the list of the
href values where the conversion failed, and the list of the href values outside
the url folder. If the second list is empty, you can read 'No errors' instead of
the empty list.

'pdf_options' is a dictionary with the options for pdfkit, see the documentation
of pdfkit.

Requirements: generate_log, get_internal_and_external_hrefs, get_url_dir, os,
os.path, pdfkit, url2log, and url2pdf

:param url: A string
:param output_dir: A string
:param pdf_options: A dictionary, by default DEFAULT_PDF_OPTIONS
:param log_filename: A string, by default DEFAULT_LOG
:return: None
"""

# CONSTANTS:
INFO = 0  # To use 'url_href_info'
PDFS = 1  # To use 'generate_pdfs'

# INPUTS:
# Function to run:
function = PDFS

# The input url:
input_url = 'ManualURL'
# The  output directory (only for function PDFS):
output_dir = 'output\\ManualName'

# pdfkit options (only for function PDFS):
options = {
    'page-size': 'Letter',
    # 'orientation': 'Landscape',
    # 'no-outline': None
    }
```

```
# Log filename (only for function PDFS):
log_filename = 'web_manual2pdf.log'

if __name__ == '__main__':
    if function == INFO:
        wmp.url_href_info(input_url)
    if function == PDFS:
        wmp.generate_pdfs(input_url,
                          output_dir,
                          options,
                          log_filename)
```

3 The script rename_dir_files.py

3.1 How it works

There are web manuals whose web pages are all default files, e.g. Flask manual[1]. In those cases, when we use the script `web_manual2pdf.py` to download the manual as a set of PDF files, all these files have the same name, `[default].pdf`. These PDF files are evidently in different folders, one by folder. With this script, we can rename each file `path\\[default].pdf` inside the manual local folder. The new filename (with path) could be `path.pdf`, for example.

This script is an interface for the function `rename_dir_files`. This is a function from the module `file_and_dir_utils.py`.

This script needs values for the following four variables

- `base_dir`, a string. It must be a valid directory name
- `old_str`, a string. It must be part of a valid path name of a PDF file, typically `\\[default.pdf]` when it is used with the output of `generate_pdfs`
- `new_str`, a string. It must be part of a valid path name of a PDF file, typically `.pdf` when it is used with the output of `generate_pdfs`
- `if_alone`, boolean (`True` or `False`)

This script changes the path of some files in the directory `base_dir`. For simplicity, we assume first that the variable `if_alone` value is `False`. The files whose path is changed are those whose path string contains `old_str`, the new path is also inside `base_dir`, and there is no file with the new path. The new path is the original path with `old_str` replaced by `new_str`.

If `if_alone` is `True`, then only the files that are alone in a subdirectory can be renamed. Here a file is alone if no other file is in the same subdirectory. Note that the files `a/b/file_1.pdf` and `a/b/c/file_2.pdf` are in different subdirectories.

After renaming those files the function deletes the empty subdirectories of `base_dir`. Here, a directory is empty if it and its subdirectories do not contain files.

[1] https://flask.palletsprojects.com/en/2.2.x/

Example 1. Suppose that we have the directory `C:\example` with the following files:

```
[default].pdf
a.pdf
b.txt
a\[default].pdf
b\[default].pdf
b\x.pdf
b\c\[default].pdf
```

Assume also that this folder does not contain empty subfolders.

If we run the script `rename_dir_files.py` with the variable values

```
base_dir = "C:\\example"
old_str = "\\[default].pdf"
new_str = ".pdf"
if_alone = True
```

then the files are renamed to

```
[default].pdf
a.pdf
b.txt
a\[default].pdf
b\[default].pdf
b\c.pdf
b\x.pdf
```

If we run the script `rename_dir_files.py` with the variable values

```
base_dir = "C:\\example"
old_str = "\\[default].pdf"
new_str = ".pdf"
if_alone = False
```

then the files are renamed to

```
[default].pdf
a.pdf
b.pdf
b.txt
a\[default].pdf
b\c.pdf
b\x.pdf
```

In both cases, the empty subfolder `b\c` is deleted, and we obtain the following output in the terminal:

```
The file C:\example\a.pdf
already exists. So, we cannot rename the file
C:\example\a\[default].pdf
```

3.2 The script

The script is the following

```
import file_and_dir_utils as fdu

"""
This script is an interface for the function 'rename_dir_files' from
file_and_dir_utils.py.

DOCSTRING OF RENAME_DIR_FILES:
This function changes the path of some files in the directory 'base_dir'.
First we assume that the value of the fourth parameter, 'if_alone', is
False, the default value. The files whose path is changed are those whose
path string contains 'old_str' and the new path is also inside 'base_dir'.
The new path is the original path with 'old_str' replaced by 'new_str'.

If 'if_alone' is True, then only the files that are alone in a subdirectory
can be renamed. Here a file is alone if no other file is in the same
subdirectory. Note: the files 'a/b/file_1.pdf' and 'a/b/c/file_2.pdf' are
in different subdirectories.

If there exists already a file with the new filename, the file is not renamed,
and a message is displayed on the terminal.

After renaming those files the function deletes the empty subdirectories of
'base_dir'. Here a directory is empty if it and its subdirectories do not
contain files.

Note: If there are only two files in

Requirements: os and del_empty_subdirs

Required by: rename_dir_files script

:param base_dir: A string
:param old_str: A string
:param new_str: A string
:param if_alone: Boolean, by default False
:return: None
"""

# ARGUMENTS
base_dir = "output\\ManualName"
old_str = "\\[default].pdf"
new_str = ".pdf"
```

```
if_alone = True

if __name__ == '__main__':
    fdu.rename_dir_files(base_dir, old_str, new_str, if_alone)
```

4 Installation

You need Python 3.9. Download the project folder `web_manual2pdf`, create a virtual environment with `pip` and `requirements.txt`, and run with this virtual environment the script that you want. To uninstall it, simply delete the folders of the project and the virtual environment.