# MDtoLongPDF

**Table of Contents**



# About The Project

Pagination in PDF sucks. The vast majority of PDF documents nowadays were never intended to, nor will ever be printed. Yet, page breaks continue to split sections, break tables, move figures around leaving huge chunks of empty space, etc, all to serve a function that is no longer needed.

So, this script aims to provide a workaround when converting from inherently unpaginated formats, namely, Markdown and HTML, into PDF by putting all of their rendered content onto a single very long PDF page. See README.pdf for an example.

# Getting Started

## Prerequisites

- Python 3.9+
  - `numpy`
  - `pdfminer`
  - (optional) `nbconvert` & `nbformat` for ipynb -> md
- `pandoc` for md -> html
- `PrinceXML` for html -> pdf
- (optional) `qpdf` to remove the watermark of free `PrinceXML` version - extra convenience but can be easily removed manually or by purchasing the license. See Removing PrinceXML Watermark section for caveats.

## Installation

1. Install `pandoc`, `Prince XML`, and `qpdf`
2. Clone the repo
   `sh git clone https://github.com/Breedoon/MDtoLongPDF.git`
3. Create a virtual environment and activate it:

```
python3 -m venv ./venv
source venv/bin/activate
```

4. Install `requirements.txt`:

```
pip install -r requirements.txt
```

# Usage

There are two ways to run the script: with command-line arguments, and with arguments from Python inputs.

## Running from Command-line

The usage of the script from command line is as follows:

```
./md2longpdf.py [in_file] [-h] [--output-path=OUT_PATH]

positional arguments:
  in_file               absolute path to the md/html/ipynb file

optional arguments:
  -h, --help            show this help message and exit
  --output-path=OUT_PATH, -o OUT_PATH
                        absolute path to directory where to put the produced PDF file
```

For example:

```
./md2longpdf.py README.md -o resources
```

```
Running FetchFile...
Running HTMLtoPDF...
Running RemovePrinceWatermark...
Running ReturnFile...
PDF generated into /Users/breedoon/MDtoLongPDF/resources/README.pdf
```

If you do not have `qpdf` installed to remove the `PxinceXML` watermark, the output file will still be generated, just with the watermark.

### Additional parameters

You can also specify additional parameters that will be passed to the execution modules:

```
./md2longpdf.py README.md -o resources --margin_bottom_mm=0 --page_width_mm=210
```

## Running as a Python program

1. First just run:

```
./md2longpdf.py
```

2. Then you will be prompted to enter path to you `.md`, `.html`, or `.ipynb` file, for example:

```
Enter location of your md/html/ipynb file (e.g.: /Users/breedoon/
CS166/Assignment 1.md)
>? README.md
```

3. Then you will be asked where the PDF should be generated (or leave empty to use the same folder and name as the input file):

```
Enter location to store the produced PDF file (e.g.: /Users/breedoon/
CS166/PDF/)
or leave blank to generate in the same folder
>? resources
```

4. Done! Find your PDF: resources/README.pdf

If you do not have `qpdf` installed to remove the `PxinceXML` watermark, the output file will still be generated, just with the watermark.

## Default Parameters

The stylesheet used to generate HTML is resources/pandoc.css based on this stylesheet. The simplest way to change scale is to modify `html { font-size: 120%; ... }`.

# Notes

## Removing PrinceXML Watermark

A free version of PrinceXML leaves a watermark in the top right corner of the produced PDF. While this watermark can easily be removed manually (or by purchasing a PrinceXML license, duh), I thought it would be useful to add the ability to remove it automatically at the end of the process.

Doing it did not prove easy using as little external tools as possible. `qpdf` (or potentially `pdftk`) would be necessary anyway to (un)compress the PDF to edit the watermark out. However, existing fixes (like this one or this one) always ended up either not working, damaging the PDF, or producing a PDF without the watermark or any other annotations including all hyperlinks within the document, which was too high of a cost. After some inquiry into PDF format, it turned out that the easiest modification to a PDF that would not damage the file should replace some content with the content of identical length (e.g., `/AP 0 41` would need to be `abcdefgh` - any 8 characters) because otherwise XREF gets damaged, and it's not very easy to repair it (`qpdf` and `Ghostrcript` did not handle it). So, after some experimentation, I found that if the coordinates of the visible box (/Rect) are to ones that don't make sense (like negative ones), the box visible part of the note disappears. So, I decided to just change the x-coordinate (which should be the same for all documents of the default width, 210mm) to a negative value.

This is a not very generalizable solution, as it works only on unix-like systems (which have `sed` command), only with pages of A4 width (210mm), requires an extra command-line tool to set up (qpdf), and even then the watermark just becomes invisible rather than disappears, and can still be found through random clicking. But this was good enough to solve my problem and was not worth spending many more hours trying to generalize it.

## Wikilinks

At the moment of writing, pandoc still does not have support for markdown [[wikilinks]] so I had to write a short script of my own to preprocess my markdown files containing wikilinks, which might not work in some edge cases (e.g., double [[]] inside of code blocks, etc), though it has worked well for most of my use cases.

`pandoc` will likely soon implement wikilinks, after which the entire `MdToMdWithoutWikilinks` part of the script will be unnecessary and the pandoc command will need to be modified to include `pandoc ... -f markdown+wikilinks_title_after_pipe ...`.

## IPYNB Conversion to Markdown

While ipynb can be converted directly to HTML (instead of ipynb -> md -> html as implemented here), I decided to make it convert to markdown first to make sure all the styles fine-tuned to work well with markdown are applied to ensure the PDF is generates correctly without, eg, code overflows or unrendered equations.