Contents

TODO:

[] write script to filter all `pandoc` templates
+ write docs to describe JSON format for templates
[] write script to create new `pandoc` template
+ [1] enter template name
+ [2] enter template command
[] write script to filter `pandoc` default settings
+ def settings saved to storage on configuration

FLOW:

- File Action on selected file
- save path to cache
- launch `dr:in` filter with format as query
- search for input format
- save format to cache
- launch `dr:out` filter
- search for output format
- save output to cache
- launch `dr:opt` filter
- search thru Boolean and Argument options
- IF BOOLEAN OPTION:
  - flip Boolean value and save to cache
- IF ARGUMENT OPTION:
  - launch `dr:opt:set` to ask for value of option.
  - save `--flag=value` to cache
- IF NOT FINAL SELECTION:
  - `rerun.py "pandoc_options"` to see/change more options
- IF FINAL SELECTION:
  - run `pandoc` with all options

FILTERS:

- `dr:in` filters all possible input formats
- `dr:out` filters all possible output formats
- `dr:opt` filters all boolean and argument options
- `dr:tmp` filters all saved `pandoc` templates [TODO]
- `dr:def` filters all default `pandoc` settings [TODO]

ACTIONS:

- File Action for new conversion
- File Action for template conversion [TODO]
- Script Actions to save filter selection to cache
- Script Action to convert file via `pandoc`

PROCESS:

1. Select file via Alfred
2. Select File Action (new)
3. Save file path to cache
    - get file extension and use are `{query}` for filter
4. Launch inputs filter with `{query}`
5. Save input to cache
6. Launch outputs filter
7. Save output to cache
8. Launch boolean options filters
9. Save all options to cache
10. Launch argument options filters
11. Save all options to cache
12. Run `pandoc` with all cached commands

[TODO] 1. Select file via Alfred 2. Select File Action (template) 3. Save file path to cache - get file extension and use are `{query}` for filter 4. Launch inputs filter with `{query}` 5. Save input to cache 6. Launch templates filter 7. Run `pandoc` with template command

```
__usage__ = """
pandoctor.py <action> [<key> | <profile>] [<query>]

PanDoctor -- An Alfred GUI for `pandoc`

Usage:
    pandoctor.py store <key> <value>
    pandoctor.py search <query>
    pandoctor.py launch <query>

Arguments:
    <key>       Dictionary key to save <value> data under in cache
    <value>     Data to be saved in cache
    <query>     Search query

Options:
    -h, --help  Show this message

This script is meant to be called from Alfred.

"""
```

FUNCTIONS:

- store
  - save data to `.cache` file
- search
  - filter data via Alfred interface
- launch
  - launch another action

1. Store API: `store(name, data)` `store()` updates (i.e. add to) `name` with the `data` * `name` = `run` or `pandoc` * `data` = - input path (`in_path`) + path to file chosen in File Action - input format (`in_fmt`) + pandoc format of input file - output format (`out_fmt`) + pandoc format of final output file - options (`options`) + any pandoc CLI options (for `pandoc`) - all inputs, outputs, and options + dicts of all pandoc info

2. Search API: `search(name, query)` `search()` filters `name` data using `query`

    - `name` =
        - inputs
        - outputs
        - options
        - templates
        - defaults
    - `query` = ANYTHING

3. Launch API: `launch(trigger, query)` `launch()` activates the `trigger` Alfred process with `query` - input search - output search - options search - arg option set - pandoc itself