

Universal Reddit Scraper - A comprehensive Reddit scraping command-line tool written in Python.

Python no status Rust no status coverage 86% release v3.4.0 tokei no longer available
license MIT

```
[-h]
[-e]
[-v]

[-t [<optional_date>]]
[--check]

[-r <subreddit> <(h|n|c|t|r|s)> <n_results_or_keywords>
[<optional_time_filter>]]
    [-y]
    [--csv]
    [--rules]
[-u <redditor> <n_results>]
[-c <submission_url> <n_results>]
    [--raw]
[-b]
    [--csv]

[-lr <subreddit>]
[-lu <redditor>]

    [--nosave]
    [--stream-submissions]

[-f <file_path>]
    [--csv]
[-wc <file_path> [<optional_export_format>]]
    [--nosave]
```

Introduction

This is a comprehensive Reddit scraping tool that integrates multiple features:

- Scrape Reddit via **PRAW** (the official Python Reddit API Wrapper)
 - Scrape Subreddits
 - Scrape Redditors
 - Scrape submission comments
- Livestream Reddit via **PRAW**
 - Livestream comments submitted within Subreddits or by Redditors
 - Livestream submissions submitted within Subreddits or by Redditors
- Analytical tools for scraped data
 - Generate frequencies for words that are found in submission titles, bodies, and/or comments
 - Generate a wordcloud from scrape results

NOTE: Requires Python 3.11+ and Poetry installed on your system.

Run the following commands to install URS :

```
git clone --depth=1 https://github.com/JosephLai241/URS.git
cd URS
poetry install
poetry shell
maturin develop --release
```

TIP: If poetry shell does not activate the virtual environment created by Poetry , run the following command to activate it:

```
source .venv/bin/activate
```

Exporting

Table of Contents

- [Export File Format](#)
 - [Exporting to CSV](#)
- [Export Directory Structure](#)
 - [PRAW Scrapers](#)
 - [PRAW Livestream Scrapers](#)
 - [Analytical Tools](#)
 - [Example Directory Structure](#)

Export File Format

All files except for those generated by the wordcloud tool are exported to JSON by default. Wordcloud files are exported to PNG by default.

`URS` supports exporting to CSV as well, but JSON is the more versatile option.

Exporting to CSV

You will have to include the `--csv` flag to export to CSV.

You can only export to CSV when using:

- The Subreddit scrapers.
- The word frequencies generator.

These tools are also suitable for CSV format and are optimized to do so if you want to use that format instead.

The `--csv` flag is ignored if it is present while using any of the other scrapers.

Export Directory Structure

All exported files are saved within the `scrapes` directory and stored in a sub-directory labeled with the date. Many more sub-directories may be created in the date directory. Sub-

directories are only created when its respective tool is run. For example, if you only use the Subreddit scraper, only the `subreddits` directory is created.

PRAW Scrapers

The `subreddits`, `redditors`, or `comments` directories may be created.

PRAW Livestream Scrapers

The `livestream` directory is created when you run any of the livestream scrapers. Within it, the `subreddits` or `redditors` directories may be created.

Analytical Tools

The `analytics` directory is created when you run any of the analytical tools. Within it, the `frequencies` or `wordclouds` directories may be created. See the [Analytical Tools](#) section for more information.

Example Directory Structure

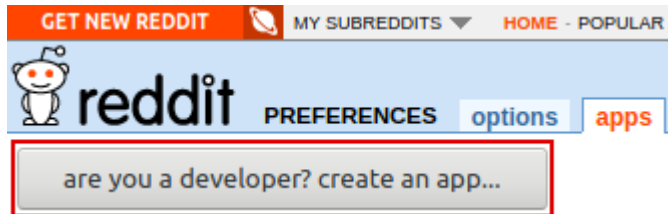
This is the [samples](#) directory structure generated by `nomad`, a modern `tree` alternative I wrote in [Rust](#).

```
scrapes/
├── 06-02-2021
│   ├── analytics
│   │   ├── frequencies
│   │   │   ├── comments
│   │   │   │   └── What's something from the 90s you miss_-all.json
│   │   │   ├── livestream
│   │   │   │   └── subreddits
│   │   │   │       └── askreddit-comments-20_44_11-00_01_10.json
│   │   │   └── subreddits
│   │   │       └── cscareerquestions-search-'job'-past-year-rules.json
│   ├── wordcloud
│   │   ├── comments
│   │   │   └── What's something from the 90s you miss_-all.png
│   │   ├── livestream
│   │   │   └── subreddits
│   │   │       └── askreddit-comments-20_44_11-00_01_10.png
│   │   └── subreddits
│   │       └── cscareerquestions-search-'job'-past-year-rules.png
│   ├── comments
│   │   └── What's something from the 90s you miss_-all.json
│   ├── livestream
│   │   └── subreddits
│   │       ├── askreddit-comments-20_44_11-00_01_10.json
│   │       └── askreddit-submissions-20_46_12-00_01_52.json
│   ├── redditors
│   │   └── spez-5-results.json
│   ├── subreddits
│   │   ├── askreddit-hot-10-results.json
│   │   └── cscareerquestions-search-'job'-past-year-rules.json
│   └── urs.log
```

How to Get PRAW Credentials

Create your own Reddit account and then head over to [Reddit's apps page](#).

Click "are you a developer? create an app...".



Name your app, choose "script" for the type of app, and type "http://localhost:8080" in the redirect URI field since this is a personal use app. You can also add a description and an about URL.

create application

Please [read the API usage guidelines](#) before creating your application. After creating, you will be required to [register](#) for production API use.

A screenshot of the 'create application' form on Reddit. The form has several fields: 'name' (containing 'Reddit Scraper'), 'type' (with radio buttons for 'web app', 'installed app', and 'script', where 'script' is selected), 'description' (containing 'Scrape Subreddits, Redditors, and post comments.'), 'about url' (empty), and 'redirect uri' (containing 'http://localhost:8080'). Each of these fields is highlighted with a red rectangular box. At the bottom of the form is a 'create app' button.

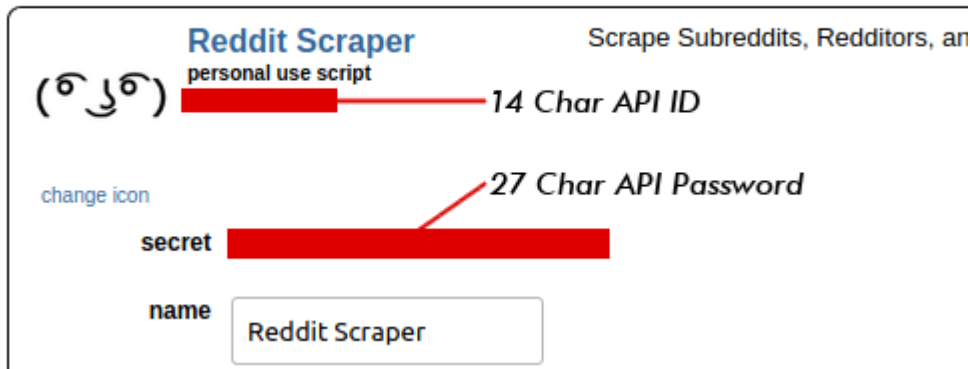
Click "create app", then "edit" to reveal more information.

developed applications



You should see a string of 14 characters on the top left corner underneath "personal use script". That is your API ID. Further down you will see "secret" and a string of 27 characters; that is your API password. **Save this information as it will be used in the program in order to access the Reddit API.**

developed applications



You will also have to provide your app name and Reddit account username and password in the block of credentials found in `.env`.

Scrape Speeds

Your internet connection speed is the primary bottleneck that will establish the scrape duration; however, there are additional bottlenecks such as:

- The number of results returned for Subreddit or Redditor scraping.
- The submission's popularity (total number of comments) for submission comments scraping.

Rate Limits

Yes, PRAW has rate limits. These limits are proportional to how much karma you have accumulated -- the higher the karma, the higher the rate limit. This has been implemented to mitigate spammers and bots that utilize PRAW.

Rate limit information for your account is displayed in a small table underneath the successful login message each time you run any of the PRAW scrapers. I have also added a `--check flag` if you want to quickly view this information.

`URS` will display an error message as well as the rate limit reset date if you have used all your available requests.

There are a couple ways to circumvent rate limits:

- Scrape intermittently
- Use an account with high karma to get your PRAW credentials
- Scrape less results per run

Available requests are refilled if you use the PRAW scrapers intermittently, which might be the best solution. This can be especially helpful if you have automated `URS` and are not looking at the output on each run.

Subreddit, Redditor, and Submission Comments Attributes

These attributes are included in each scrape.

Subreddits (submissions)	Redditors	Submission Comments
author	comment_karma	author
created_utc	created_utc	body
distinguished	fullname	body_html
edited	has_verified_email	created_utc
id	icon_img	distinguished
is_original_content	id	edited
is_self	is_employee	id
link_flair_text	is_friend	is_submitter
locked	is_mod	link_id
name	is_gold	parent_id
num_comments	link_karma	score
nsfw	name	stickied
permalink	subreddit	
score	* trophies	
selftext	* comments	
spoiler	* controversial	
stickied	* downvoted (may be forbidden)	
title	* gilded	
upvote_ratio	* gildings (may be forbidden)	
url	* hidden (may be forbidden)	
	* hot	
	* moderated	
	* multireddits	
	* new	
	* saved (may be forbidden)	
	* submissions	
	* top	

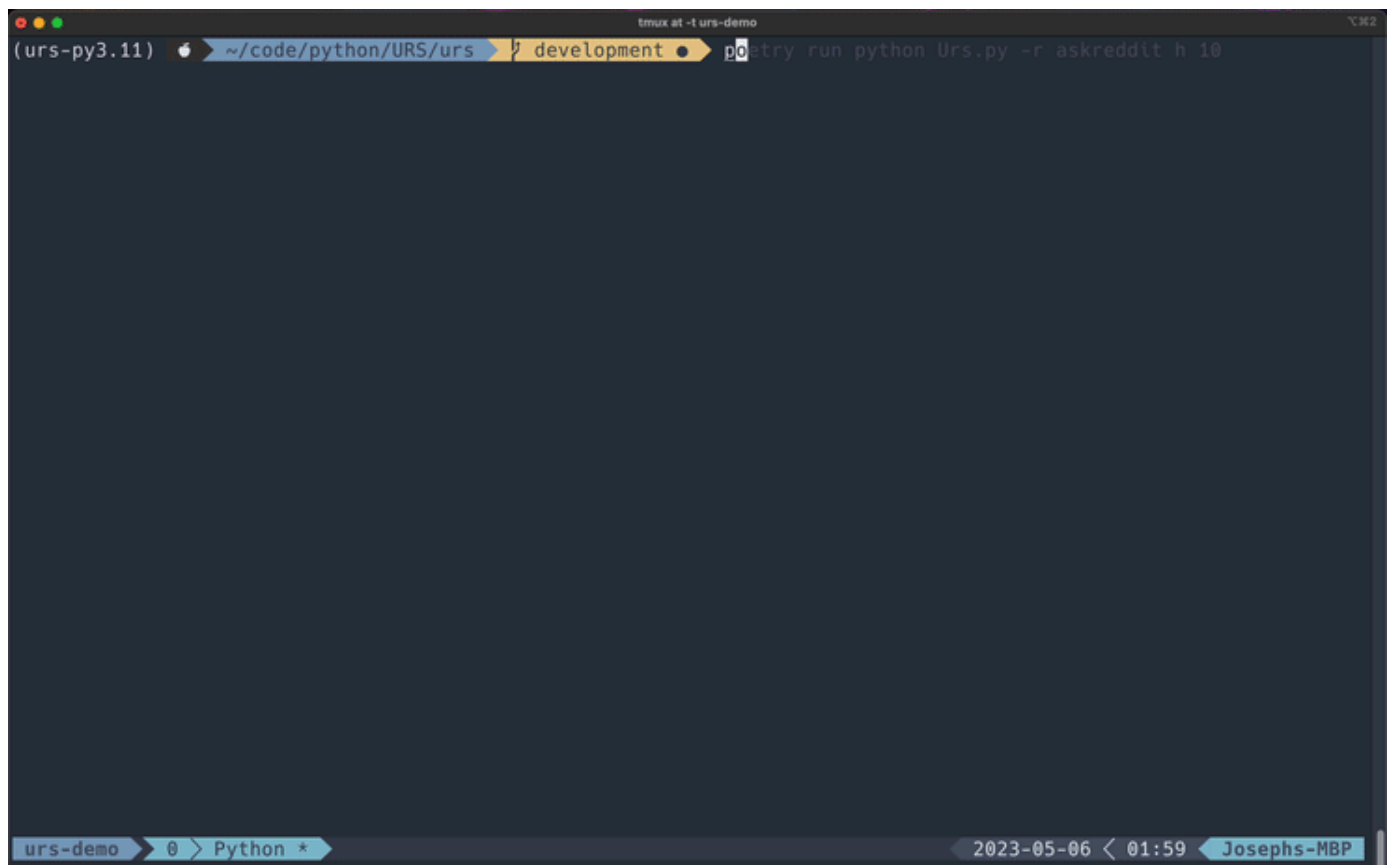
Subreddits (submissions)	Redditors	Submission Comments
	* upvoted (may be forbidden)	

**Includes additional attributes; see the [Scraping Redditors](#) section for more information.*

Table of Contents

- Subreddits
 - All Flags
 - Basic Usage
 - Filename Naming Conventions
- Time Filters
 - Filename Naming Conventions
- Subreddit Rules and Post Requirements
- Bypassing the Final Settings Check

Subreddits



All Flags

These are all the flags that may be used when scraping Subreddits.

```
[ -r <subreddit> <(h|n|c|t|r|s)> <n_results_or_keywords>  
[<optional_time_filter>]]  
    [-y]  
    [--csv]  
    [--rules]
```

Basic Usage

```
poetry run Urs.py -r <subreddit> <(h|n|c|t|r|s)> <n_results_or_keywords>
```

Supports exporting to CSV. To export to CSV, include the `--csv` flag.

Specify Subreddits, the submission category, and how many results are returned from each scrape. I have also added a search option where you can search for keywords within a Subreddit.

These are the submission categories:

- Hot
- New
- Controversial
- Top
- Rising
- Search

Filename Naming Conventions

The file names for all categories except for Search will follow this format:

```
[SUBREDDIT]-[POST_CATEGORY]-[N_RESULTS]-result(s).[FILE_FORMAT]
```

If you searched for keywords, file names will follow this format:

```
[SUBREDDIT]-Search-'[KEYWORDS]'. [FILE_FORMAT]
```

Scrape data is exported to the `subreddits` directory.

NOTE: Up to 100 results are returned if you search for keywords within a Subreddit. You will not be able to specify how many results to keep.

Time Filters

Time filters may be applied to some categories. Here is a table of the categories on which you can apply a time filter as well as the valid time filters.

Categories	Time Filters
Controversial	All (default)
Search	Day
Top	Hour
	Month
	Week
	Year

Specify the time filter after the number of results returned or keywords you want to search for:

```
poetry run Urs.py -r <subreddit> <(c|t|s)> <n_results_or_keywords>
[<time_filter>]
```

If no time filter is specified, the default time filter `all` is applied. The Subreddit settings table will display `None` for categories that do not offer the additional time filter option.

Filename Naming Conventions

If you specified a time filter, `-past-[TIME_FILTER]` will be appended to the file name before the file format like so:

```
[SUBREDDIT]-[POST_CATEGORY]-[N_RESULTS]-result(s)-past-[TIME_FILTER].
[FILE_FORMAT]
```

Or if you searched for keywords:

```
[SUBREDDIT]-Search-'[KEYWORDS]'-past-[TIME_FILTER].[FILE_FORMAT]
```

Subreddit Rules and Post Requirements

You can also include the Subreddit's rules and post requirements in your scrape data by including the `--rules` flag. **This is only compatible with JSON.** This data will be included in the `subreddit_rules` field.

If rules are included in your file, `-rules` will be appended to the end of the file name.

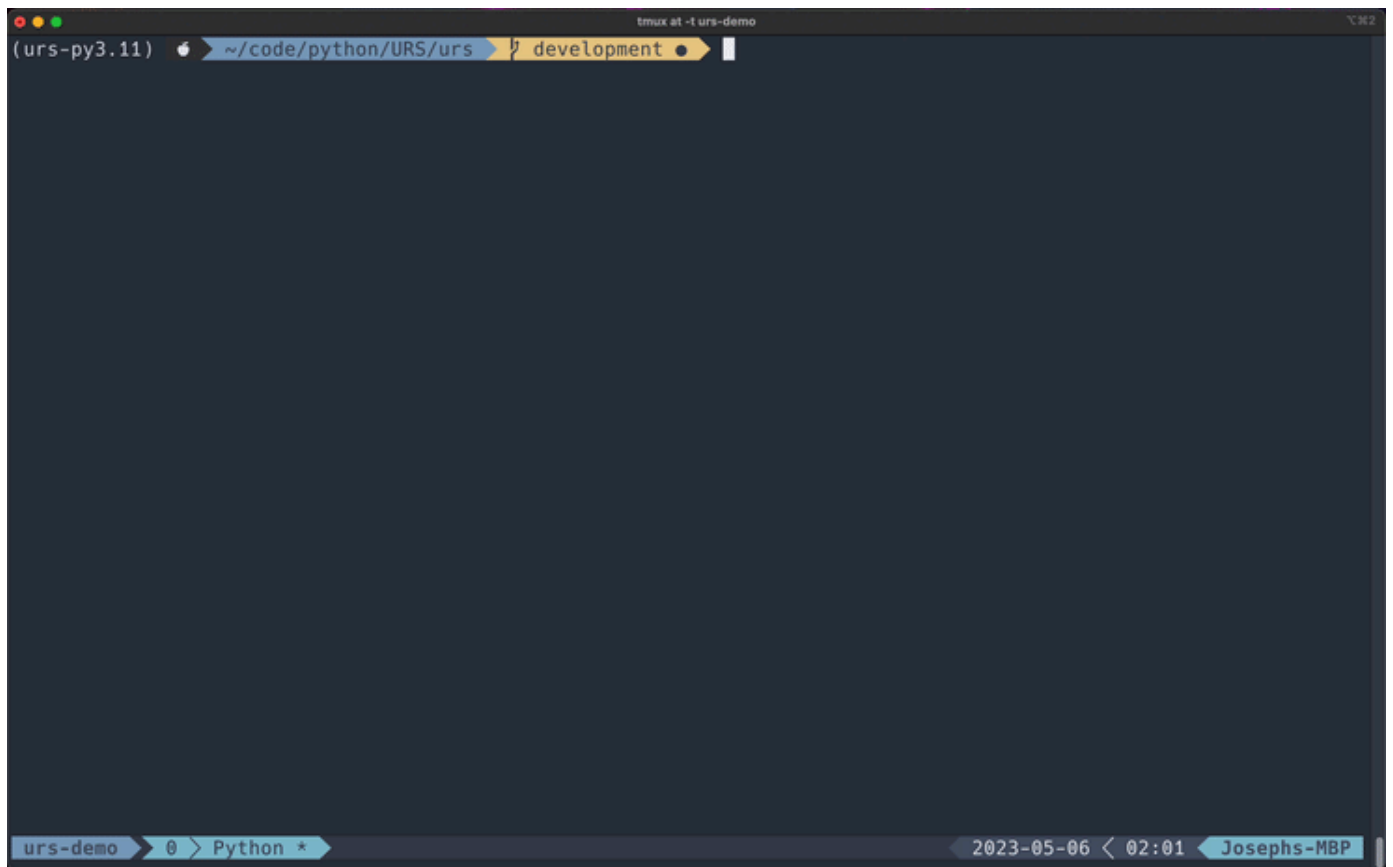
Bypassing the Final Settings Check

After submitting the arguments and Reddit validation, `URS` will display a table of Subreddit scraping settings as a final check before executing. You can include the `-y` flag to bypass this and immediately scrape.

Table of Contents

- [Redditors](#)
 - [All Flags](#)
 - [Usage](#)
 - [Redditor Interaction Attributes](#)
 - [Reddit Object Attributes](#)
 - [File Naming Conventions](#)

Redditors



**This GIF has been cut for demonstration purposes.*

NOTE: If you are not allowed to access a Redditor's lists, PRAW will raise a 403 HTTP Forbidden exception and the program will just append `"FORBIDDEN"` underneath that section in the exported file.

All Flags

These are all the flags that may be used when scraping Redditors.


```
[-u <redditor> <n_results>]
```

NOTE: The number of results returned are applied to all attributes. I have not implemented code to allow users to specify different number of results returned for individual attributes.

Usage

```
poetry run Urs.py -u <redditor> <n_results>
```

Redditor information will be included in the `information` field and includes the following attributes:

- `comment_karma`
- `created_utc`
- `fullname`
- `has_verified_email`
- `icon_img`
- `id`
- `is_employee`
- `is_friend`
- `is_mod`
- `is_gold`
- `link_karma`
- `name`
- `subreddit`
- `trophies`

Redditor Interaction Attributes

Redditor interactions will be included in the `interactions` field. Here is a table of all Redditor interaction attributes that are also included, how they are sorted, and what type of Reddit objects are included in each.

Attribute Name	Sorted By/Time Filter	Reddit Objects
Comments	Sorted By: New	Comments

Attribute Name	Sorted By/Time Filter	Reddit Objects
Controversial	Time Filter: All	Comments and submissions
Downvoted	Sorted By: New	Comments and submissions
Gilded	Sorted By: New	Comments and submissions
Gildings	Sorted By: New	Comments and submissions
Hidden	Sorted By: New	Comments and submissions
Hot	Determined by other Redditors' interactions	Comments and submissions
Moderated	N/A	Subreddits
Multireddits	N/A	Multireddits
New	Sorted By: New	Comments and submissions
Saved	Sorted By: New	Comments and submissions
Submissions	Sorted By: New	Submissions
Top	Time Filter: All	Comments and submissions
Upvoted	Sorted By: New	Comments and submissions

These attributes contain comments or submissions. Subreddit attributes are also included within both.

Reddit Object Attributes

This is a table of all attributes that are included for each Reddit object:

Subreddits	Comments	Submissions	Multireddits
can_assign_link_flair	body	author	can_edit
can_assign_user_flair	body_html	created_utc	copied_from
created_utc	created_utc	distinguished	created_utc
description	distinguished	edited	description
description_html	edited	id	description

Subreddits	Comments	Submissions	Multimedia
display_name	id	is_original_content	display_name
id	is_submitter	is_self	name
name	link_id	link_flair_text	nsfw
nsfw	parent_id	locked	subreddit_id
public_description	score	name	visibility
spoilers_enabled	stickied	num_comments	
subscribers	* submission	nsfw	
user_is_banned	subreddit_id	permalink	
user_is_moderator		score	
user_is_subscriber		selftext	
		spoiler	
		stickied	
		* subreddit	
		title	
		upvote_ratio	
		url	

* Contains additional metadata.

File Naming Conventions

The file names will follow this format:

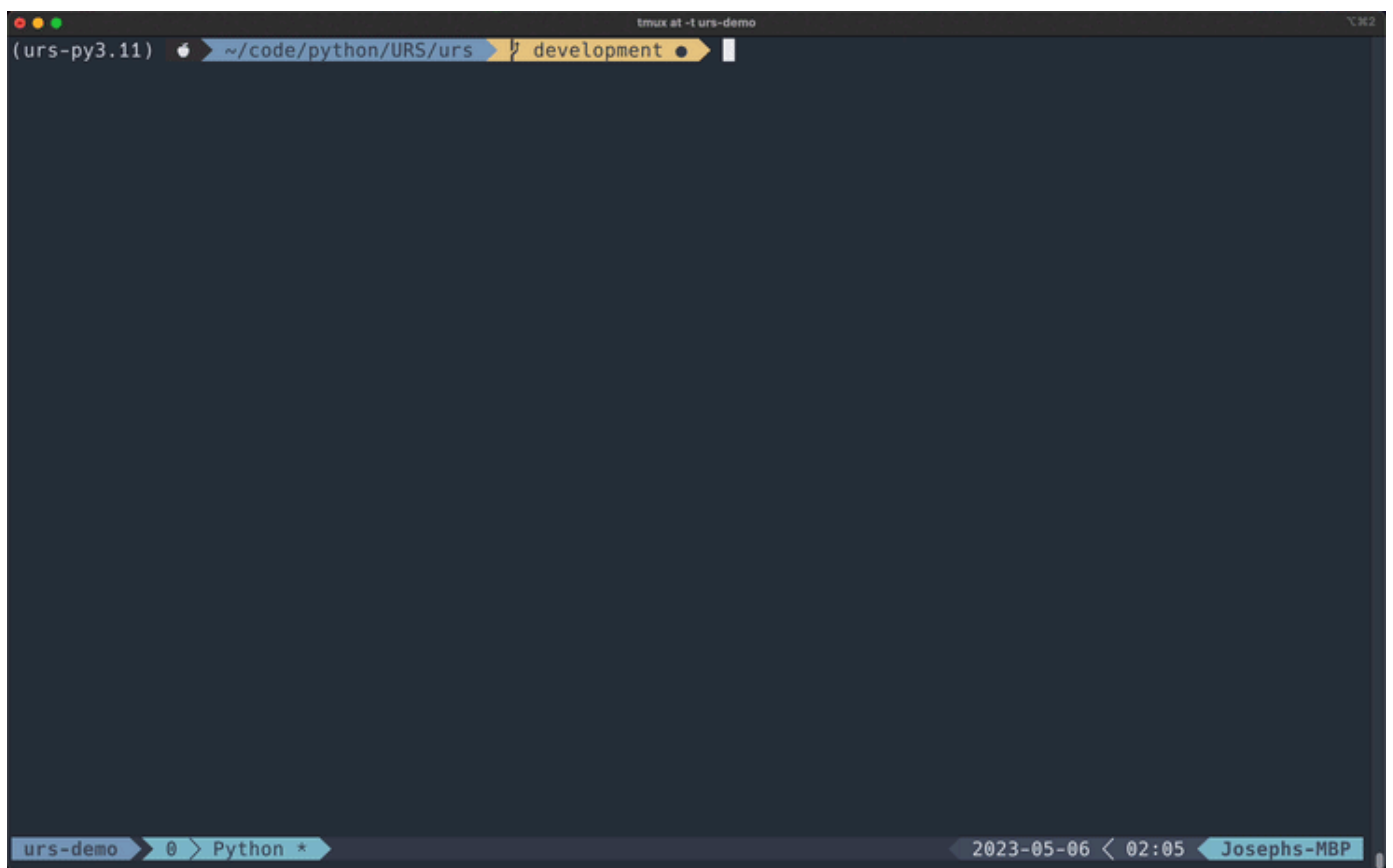
```
[USERNAME]-[N_RESULTS]-result(s).json
```

Scrape data is exported to the `redditors` directory.

Table of Contents

- [Submission Comments](#)
 - [All Flags](#)
 - [Usage](#)
 - [File Naming Conventions](#)
 - [Number of Comments Returned](#)
 - [Structured Comments](#)
 - [Raw Comments](#)

Submission Comments



**This GIF has been cut for demonstration purposes.*

All Flags

These are all the flags that may be used when scraping submission comments.

```
[-c <submission_url> <n_results>]  
[--raw]
```

Usage

```
poetry run Urs.py -c <submission_url> <n_results>
```

Submission metadata will be included in the `submission_metadata` field and includes the following attributes:

- `author`
- `created_utc`
- `distinguished`
- `edited`
- `is_original_content`
- `is_self`
- `link_flair_text`
- `locked`
- `nsfw`
- `num_comments`
- `permalink`
- `score`
- `selftext`
- `spoiler`
- `stickied`
- `subreddit`
- `title`
- `upvote_ratio`

If the submission contains a gallery, the attributes `gallery_data` and `media_metadata` will be included.

Comments are written to the `comments` field. They are sorted by "Best", which is the default sorting option when you visit a submission.

PRAW returns submission comments in level order, which means scrape speeds are proportional to the submission's popularity.

File Naming Conventions

The file names will generally follow this format:

```
[POST_TITLE]-[N_RESULTS]-result(s).json
```

Scrape data is exported to the `comments` directory.

Number of Comments Returned

You can scrape all comments from a submission by passing in `0` for `<n_results>`. Subsequently, `[N_RESULTS]-result(s)` in the file name will be replaced with `all`.

Otherwise, specify the number of results you want returned. If you passed in a specific number of results, the structured export will return up to `<n_results>` top level comments and include all of its replies.

Structured Comments

This is the default export style. Structured scrapes resemble comment threads on Reddit. This style takes just a little longer to export compared to the raw format because `URS` uses [depth-first search](#) to create the comment `Forest` after retrieving all comments from a submission.

If you want to learn more about how it works, refer to [The Forest](#), where I describe how I implemented the `Forest`, and [Speeding up Python With Rust](#) to learn about how I drastically improved the performance of the `Forest` by rewriting it in Rust.

Raw Comments

Raw scrapes do not resemble comment threads, but returns all comments on a submission in level order: all top-level comments are listed first, followed by all second-level comments, then third, etc.

You can export to raw format by including the `--raw` flag. `-raw` will also be appended to the end of the file name.

Livestreaming Reddit via PRAW

These tools may be used to livestream comments or submissions submitted within Subreddits or by Redditors.

Comments are streamed by default. To stream submissions instead, include the `--stream-submissions` flag.

New comments or submissions will continue to display within your terminal until you abort the stream using `Ctrl + C`.

File Naming Conventions

The filenames will follow this format:

```
[SUBREDDIT_OR_REDDITOR]-[comments_OR_submissions]-[START_TIME_IN_HOURS_MINUTES_SECONDS]-[DURATION_IN_HOURS_MINUTES_SECONDS].json
```

This file is saved in the main `livestream` directory into the `subreddits` or `redditors` directory depending on which stream was run.

Reddit objects will be written to this JSON file in real time. After aborting the stream, the filename will be updated with the start time and duration.

Displayed vs. Saved Attributes

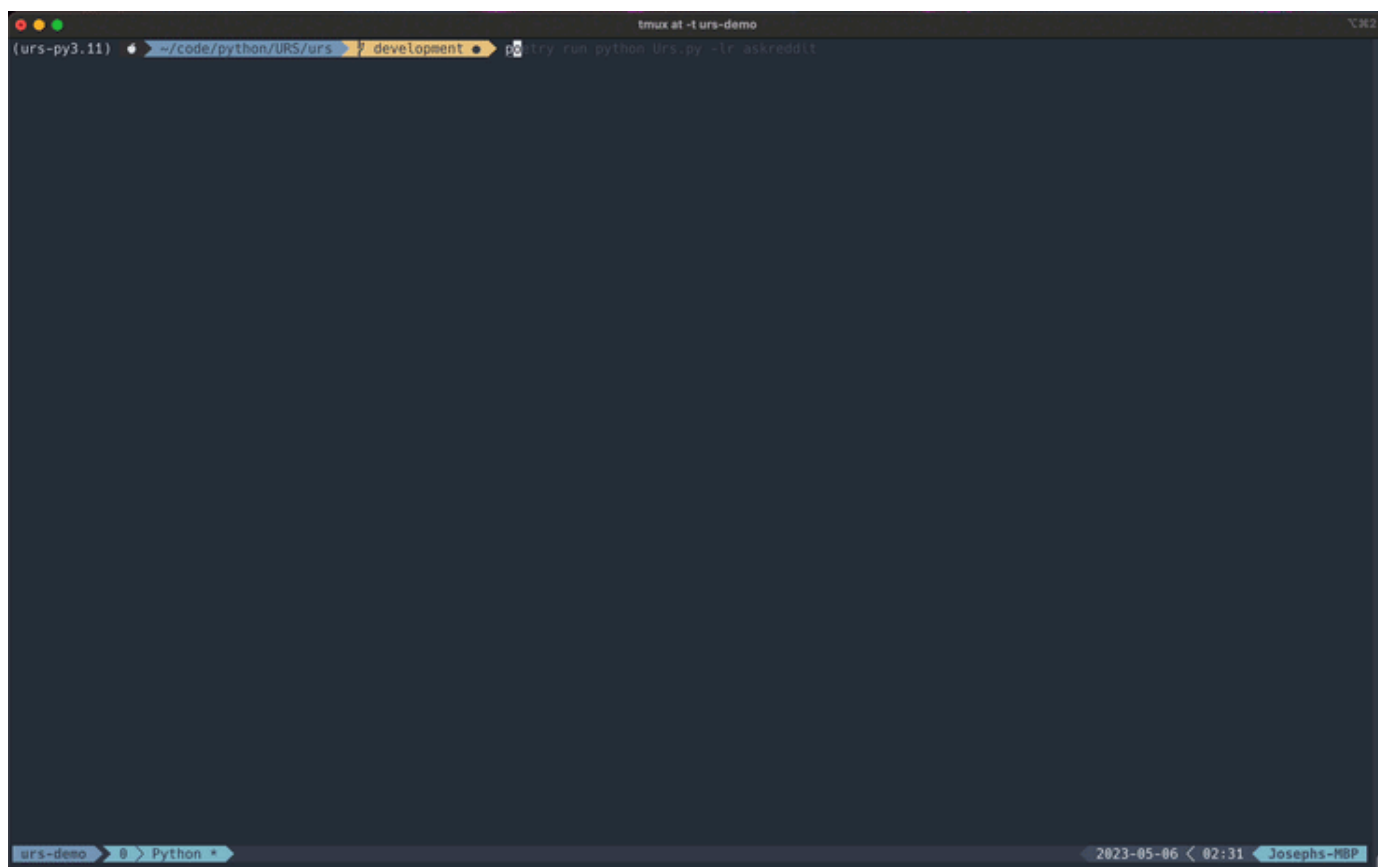
Displayed comment and submission attributes have been stripped down to essential fields to declutter the output. Here is a table of what is shown during the stream:

Comment Attributes	Submission Attributes
<code>author</code>	<code>author</code>
<code>body</code>	<code>created_utc</code>
<code>created_utc</code>	<code>is_self</code>
<code>is_submitter</code>	<code>link_flair_text</code>
<code>submission_author</code>	<code>nsfw</code>
<code>submission_created_utc</code>	<code>selftext</code>
<code>submission_link_flair_text</code>	<code>spoiler</code>
<code>submission_nsfw</code>	<code>stickied</code>

Comment Attributes	Submission Attributes
submission_num_comments	title
submission_score	url
submission_title	
submission_upvote_ratio	
submission_url	

Comment and submission attributes that are written to file will include the full list of attributes found in the [Table of All Subreddit, Redditor, and Submission Comments Attributes](#).

Livestreaming Subreddits



**This GIF has been cut for demonstration purposes.*

All Flags

These are all the flags that may be used when livestreaming Subreddits.

```
[-lr <subreddit>]
  [--nosave]
  [--stream-submissions]
```

Usage

```
poetry run Urs.py -lr <subreddit>
```

Default stream objects: Comments. To stream submissions instead, include the `--stream-submissions` flag.

You can livestream comments or submissions that are created within a Subreddit.

Reddit object information will be displayed in a [PrettyTable](#) as they are submitted.

NOTE: PRAW may not be able to catch all new submissions or comments within a high-volume Subreddit, as mentioned in [these disclaimers located in the "Note" boxes](#).

Livestreaming Redditors

Livestream demo was not recorded for Redditors because its functionality is identical to the Subreddit livestream.

All Flags

These are all the flags that may be used when livestreaming Redditors.

```
[ -lu <redditor> ]  
  [ --nosave ]  
  [ --stream-submissions ]
```

Usage

```
poetry run Urs.py -lu <redditor>
```

Default stream objects: Comments. To stream submissions instead, include the `--stream-submissions` flag.

You can livestream comments or submissions that are created by a Redditor.

Reddit object information will be displayed in a [PrettyTable](#) as they are submitted.

Do Not Save Livestream to File

Include the `--nosave` flag if you do not want to save the livestream to file.

Analytical Tools

This suite of tools can be used *after* scraping data from Reddit. Both of these tools analyze the frequencies of words found in submission titles and bodies, or comments within JSON scrape data.

There are a few ways you can quickly get the correct filepath to the scrape file:

- Drag and drop the file into the terminal.
- Partially type the path and rely on tab completion support to finish the full path for you.

Running either tool will create the `analytics` directory within the date directory. **This directory is located in the same directory in which the scrape data resides.** For example, if you run the frequencies generator on February 16th for scrape data that was captured on February 14th, `analytics` will be created in the February 14th directory. Command history will still be written in the February 16th `urs.log`.

The sub-directories `frequencies` or `wordclouds` are created in `analytics` depending on which tool is run. These directories mirror the directories in which the original scrape files reside. For example, if you run the frequencies generator on a Subreddit scrape, the directory structure will look like this:

```
analytics/  
└─ frequencies  
    └─ subreddits  
        └─ SUBREDDIT_SCRAPE.json
```

A shortened export path is displayed once `URS` has completed exporting the data, informing you where the file is saved within the `scrapes` directory. You can open `urs.log` to view the full path.

Target Fields

The data varies depending on the scraper, so these tools target different fields for each type of scrape data:

Scrape Data	Targets
Subreddit	<code>selftext</code> , <code>title</code>
Redditor	<code>selftext</code> , <code>title</code> , <code>body</code>
Submission Comments	<code>body</code>
Livestream	<code>selftext</code> and <code>title</code> , or <code>body</code>

For Subreddit scrapes, data is pulled from the `selftext` and `title` fields for each submission (submission title and body).

For Redditor scrapes, data is pulled from all three fields because both submission and comment data is returned. The `title` and `body` fields are targeted for submissions, and the `selftext` field is targeted for comments.

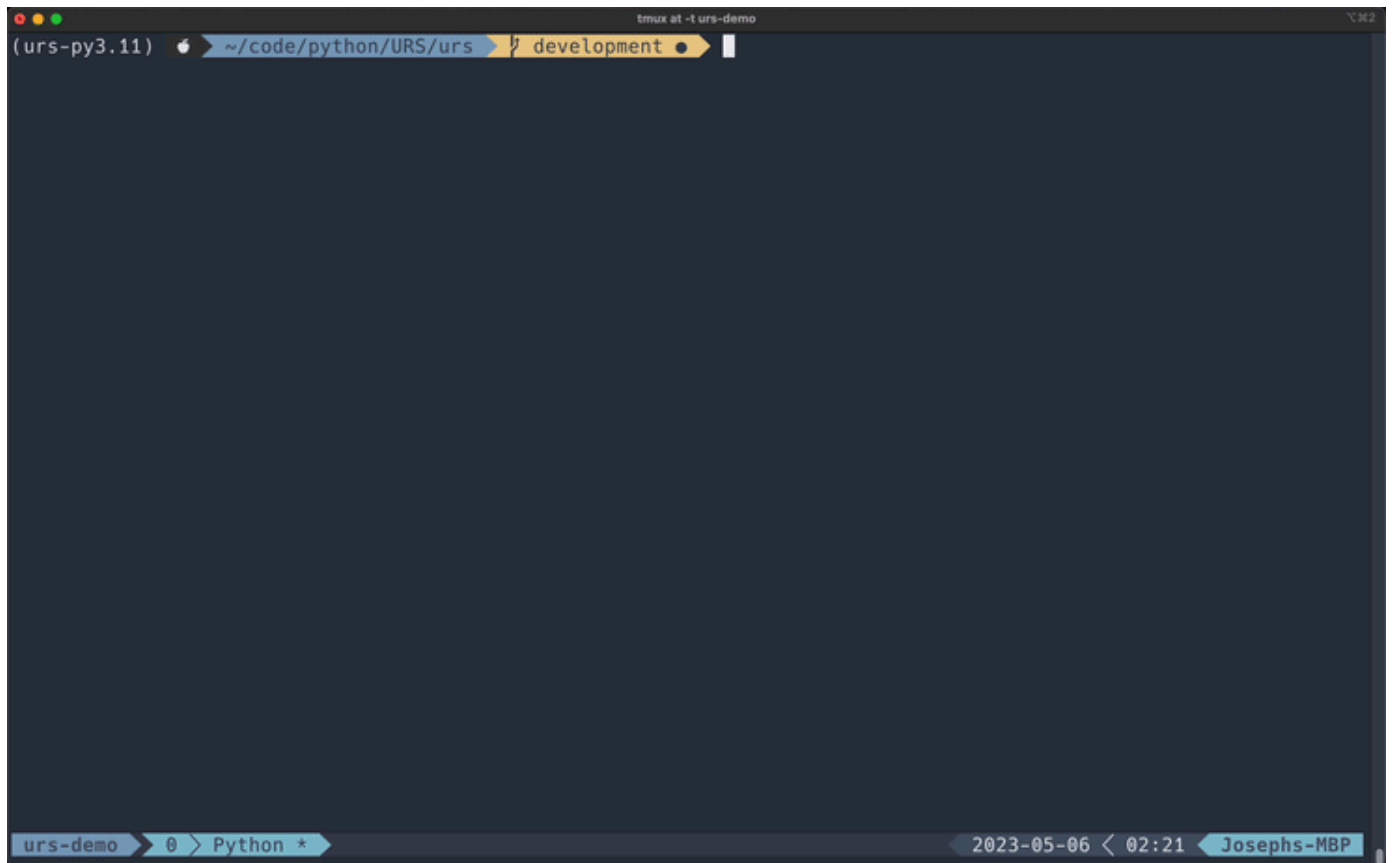
For submission comments scrapes, data is only pulled from the `body` field of each comment.

For livestream scrapes, comments or submissions may be included depending on user settings. The `selftext` and `title` fields are targeted for submissions, and the `body` field is targeted for comments.

File Names

File names are identical to the original scrape data so that it is easier to distinguish which analytical file corresponds to which scrape.

Generating Word Frequencies



All Flags

These are all the flags that may be used when generating word frequencies.

```
[-f <file_path>]
[--csv]
```

Usage

```
poetry run Urs.py -f <file_path>
```

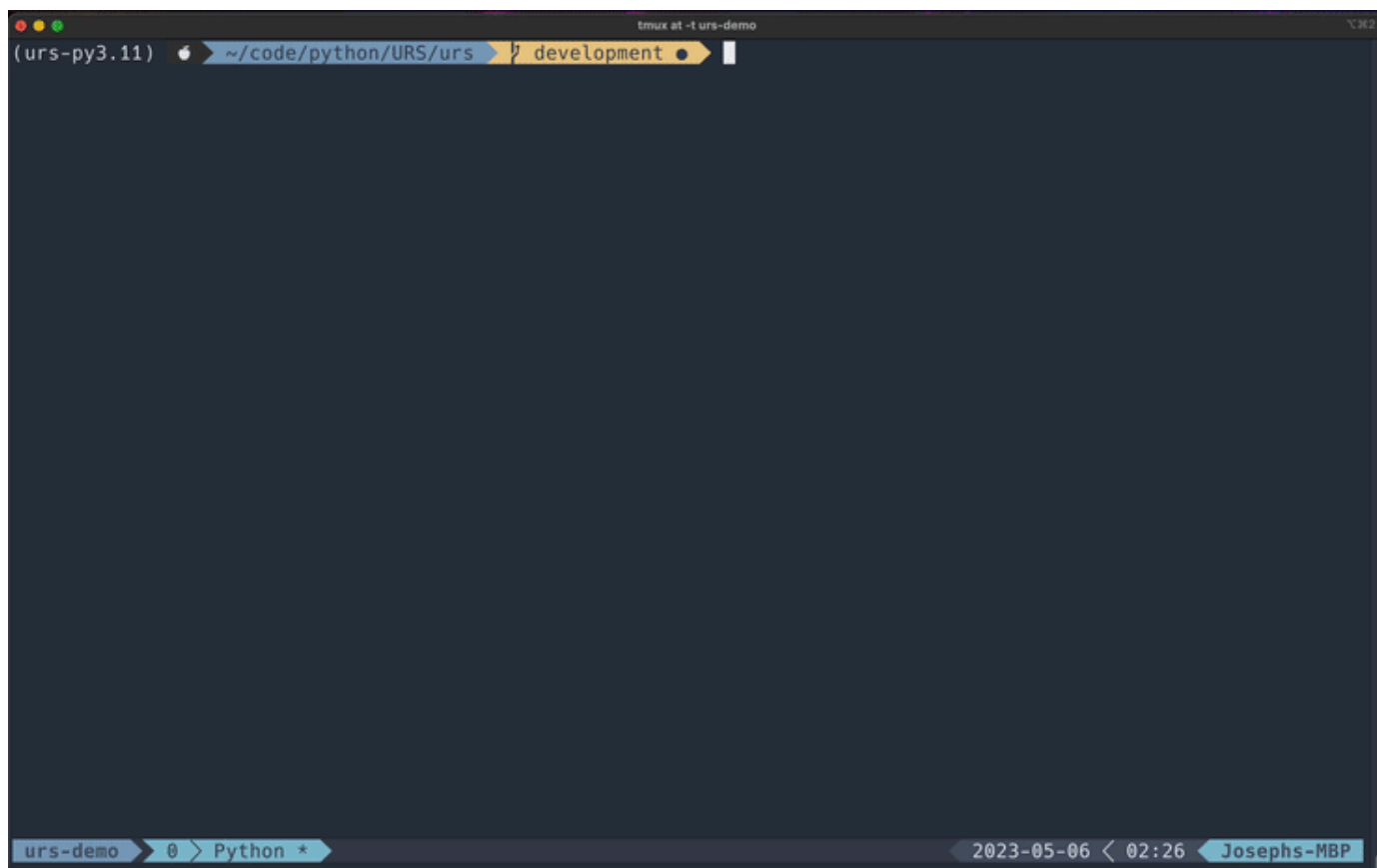
Supports exporting to CSV. To export to CSV, include the `--csv` flag.

You can generate a dictionary of word frequencies created from the words within the target fields. These frequencies are sorted from highest to lowest.

Frequencies export to JSON by default, but this tool also works well in CSV format.

Exported files will be saved to the `analytics/frequencies` directory.

Generating Wordclouds



All Flags

```
[-wc <file_path> [<optional_export_format>]]  
[--nosave]
```

Usage

```
poetry run Urs.py -wc <file_path>
```

Supported Export Formats

The following are the supported export formats for wordclouds:

- `eps`

- `jpeg`
- `jpg`
- `pdf`
- `png` (default)
- `ps`
- `rgba`
- `tif`
- `tiff`

Taking word frequencies to the next level, you can generate wordclouds based on word frequencies. This tool is independent of the frequencies generator -- you do not need to run the frequencies generator before creating a wordcloud.

PNG is the default format, but you can also export to any of the options listed above by including the format as the second flag argument.

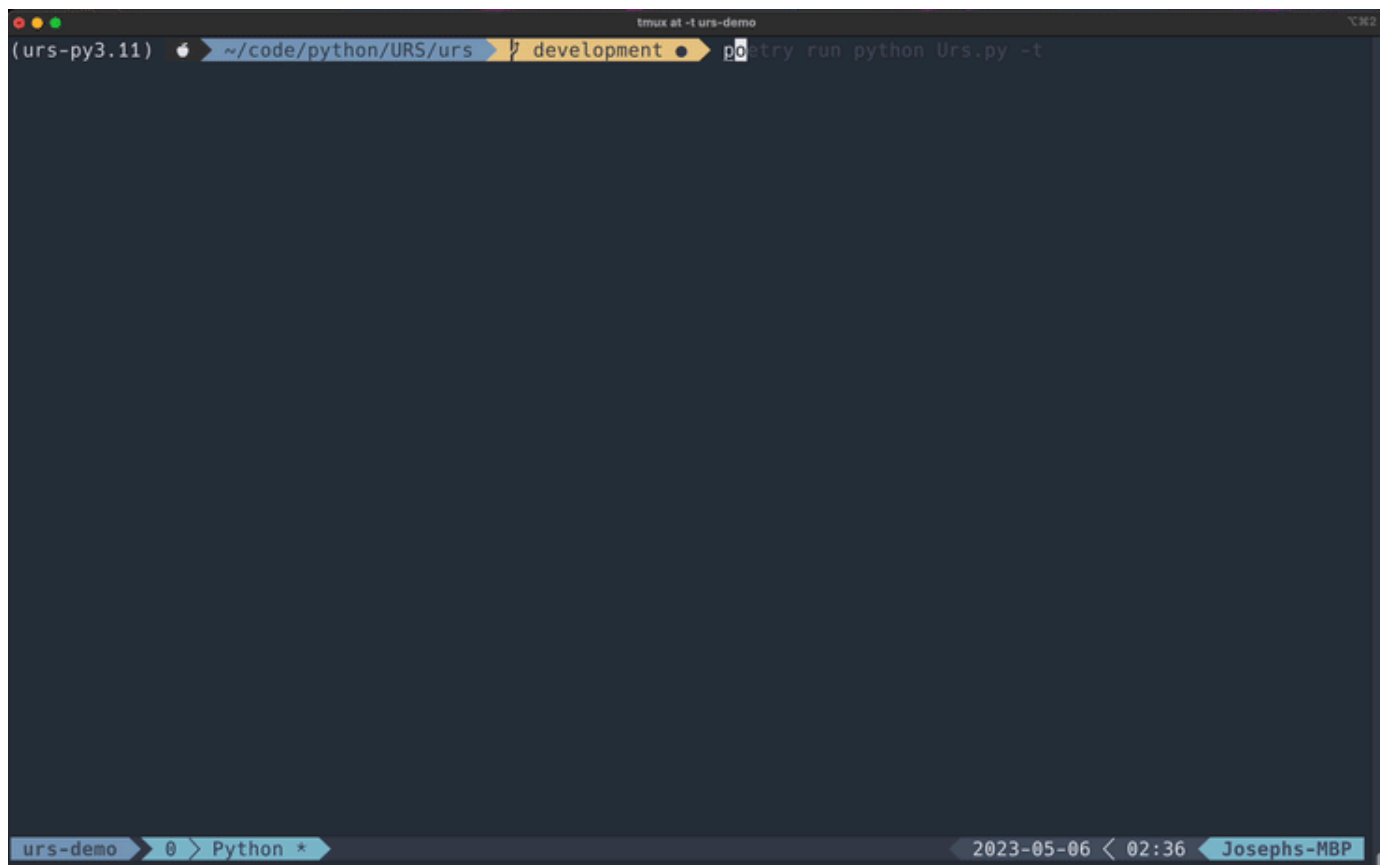
```
poetry run Urs.py -wc <file_path> [<optional_export_format>]
```

Exported files will be saved to the `analytics/wordclouds` directory.

Display Wordcloud Instead of Saving

Wordclouds are saved to file by default. If you do not want to keep a file, include the `--nosave` flag to only display the wordcloud.

Display Directory Tree



All Flags

These are all the flags that may be used when displaying the directory tree.

```
[-t [<optional_date>]]
```

Usage

If no date is provided, you can quickly view the directory structure for the current date. This is a quick alternative to `nomad` or the `tree` command.

You can also display a different day's scrapes by providing a date after the `-t` flag.

```
poetry run Urs.py -t [<optional_date>]
```

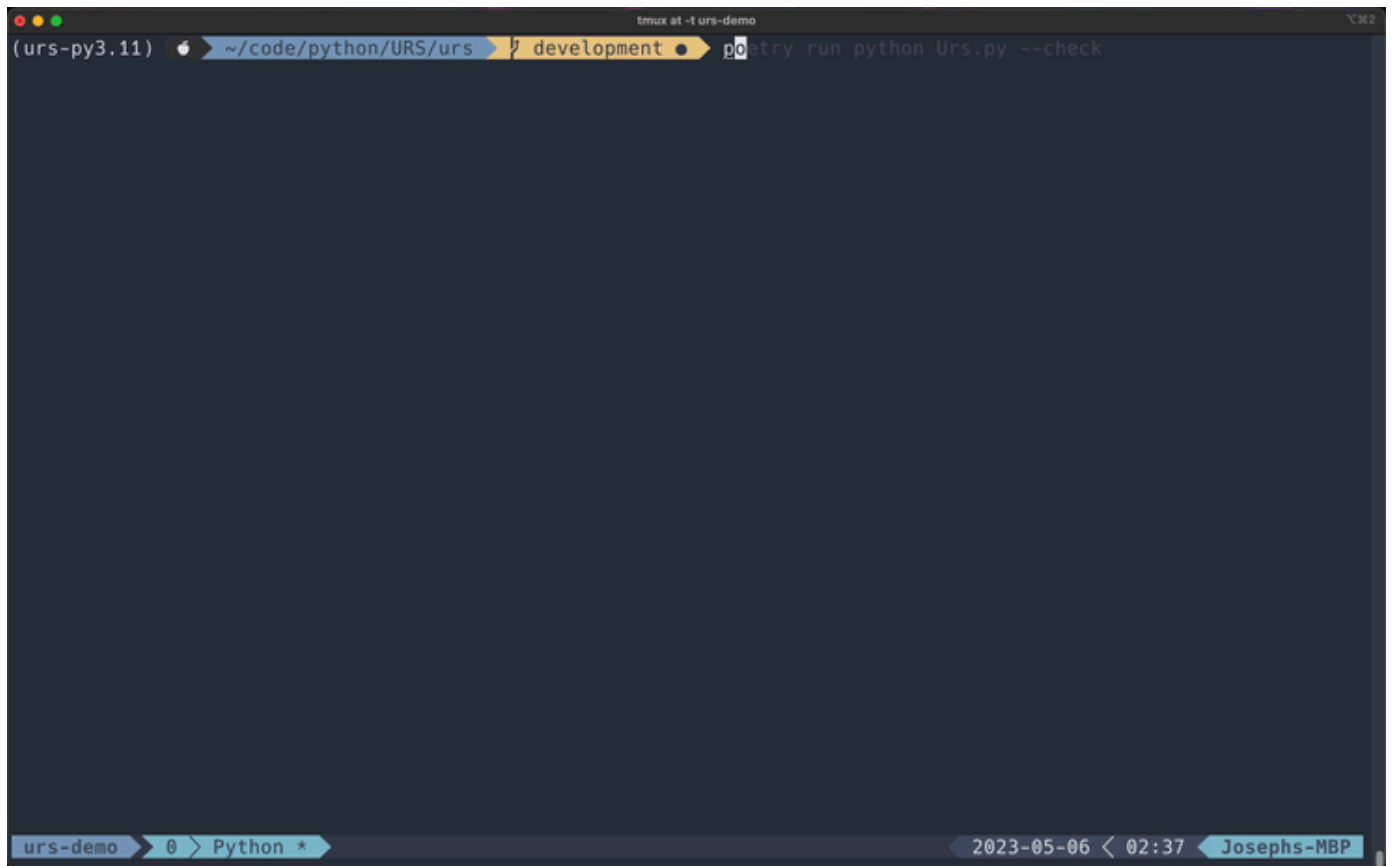
The following date formats are supported:

- `YYYY-MM-DD`

- YYYY/MM/DD

An error is displayed if `URS` was not run on the entered date (if the date directory is not found within the `scrapes/` directory).

Check PRAW Rate Limits

A screenshot of a terminal window running in tmux. The window title is 'tmux at -t urs-demo'. The prompt is '(urs-py3.11)'. The current directory is '~/code/python/URS/urs'. The branch is 'development'. The command 'poetry run python Urs.py --check' is entered. The terminal output is empty. The bottom status bar shows 'urs-demo', '0', 'Python *', '2023-05-06 < 02:37', and 'Josephs-MBP'.

You can quickly check the rate limits for your account by using this flag.

```
poetry run Urs.py --check
```

Two-Factor Authentication

If you choose to use 2FA with your Reddit account, enter your password followed by a colon and then your 2FA token in the `password` field on line 26. For example, if your password is `"p4ssw0rd"` and your 2FA token is `"123456"`, you will enter `"p4ssw0rd:123456"` in the `password` field.

2FA is NOT recommended for use with this program. This is because PRAW will raise an `OAuthException` after one hour, prompting you to refresh your 2FA token and re-enter your credentials. Additionally, this means your 2FA token would be stored alongside your Reddit username and password, which would defeat the purpose of enabling 2FA in the first place. See [here](#) for more information.

Error Messages

This document will briefly go over all the potential error messages you might run into while using URS.

Table of Contents

- Global Errors
 - Invalid Arguments
 - Export Error
- PRAW Errors
 - Invalid API Credentials or No Internet Connection
 - No Reddit Objects Left to Scrape
 - Rate Limit Reached
- Analytical Tool Errors
 - Invalid File

Global Errors

Invalid Arguments

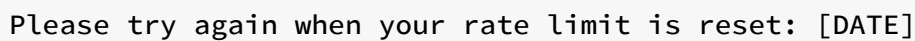
```
--\
/'--\
/\  --/
\ \____\
 \/_----/... [ERROR MESSAGE]
```

Please recheck args or refer to help for usage examples.

This message is displayed if you have entered invalid arguments. The specific error will follow

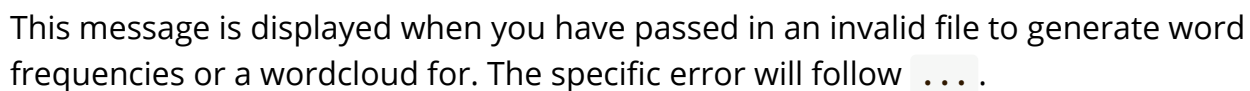
You can use the `-h` flag to see the help message or the `-e` flag to display example usage.

Rate Limit Reached



Analytical Tool Errors

Invalid File



The Forest

Created: March 17, 2021

This Python code has been deprecated as of `URS v3.4.0` and has been rewritten in Rust. However, the concepts discussed in this document as well as the implementation are still applicable to the Rust rewrite.

See [Speeding Up Python with Rust](#) for details on how I rewrote this code in Rust and how it yielded drastic performance improvements if you are interested in learning more.

Table of Contents

- [Introduction](#)
 - [Motivation](#)
 - [Inspiration](#)
- [How the Forest Works](#)
 - [The `CommentNode`](#)
 - [The `Forest`](#)
 - [The Root Node](#)
 - [How `PRAW` Comments Are Linked](#)
 - [The Insertion Methods](#)
 - [Serializing the `Forest`](#)

Introduction

Motivation

I am a self-taught software developer who just recently graduated from college and am currently looking for my first full-time job. I do not have a computer science degree, so I have had to teach myself a ton of concepts that I would have learned if I got the degree. A class I wish I was able to take in college is data structures and algorithms because that seems to be all the buzz when it comes to the technical interview, which I unfortunately struggle with greatly due to my lack of experience and practice.

Recently (March 2021) I have been teaching myself DSA. Implementing simple examples of each topic within DSA was not so bad (I am currently working on a study guide/reference repository containing these implementations in both Python and Rust that I will make public

soon), but practicing Leetcode problems was and still is a difficult process for me. I will continue to power through the struggle because my livelihood and future career depends on it, though.

While it has not been a smooth journey, I have come to realize how useful DSA is and am implementing what I have learned in a real-world use case. I do not think I would have been able to figure out a solution to the structured comments scraper's prior shortcomings if I had not studied this area within computer science. I recently implemented my first [trie](#) and was fascinated by how abstract data structures worked. I immediately realized I needed to use a tree data structure for the structured comments scraper in order to take it to the next level, which is the purpose of [this pull request](#).

Inspiration

The `Forest` is named after `PRAW`'s `CommentForest`. The `CommentForest` does not return comments in structured format, so I wrote my own implementation of it.

The trie was a huge inspiration for the `Forest`. I will quickly explain my implementation of the trie node.

```
class TrieNode():
    def __init__(self, char, is_word):
        self.char = char
        self.is_word = is_word
        self.children = dict()
```

Each node of the trie contains a character, a boolean flag indicating whether the node denotes the end of a word, and holds a dictionary filled with child nodes as values and their respective characters as keys. I could have used an array and the indices within it to emulate a dictionary, but I figured I could save some access time at the cost of extra space.

Anyways, the trie implementation is very similar to how the `Forest` works.

How the Forest Works

The `CommentNode`

I created a class `CommentNode` to store each comment's metadata and replies:


```
class CommentNode():
    def __init__(self, metadata):
        for key, value in metadata.items():
            self.__setattr__(key, value)

        self.replies = []
```

I used `__setattr__()` because the root node defers from the standard comment node schema. By using `__setattr__()`, `CommentNode` attributes will be dynamically set based on the `metadata` dictionary that has been passed in. `self.replies` holds additional `CommentNode`s.

The Forest

Next, I created a class `Forest` which holds the root node and includes methods for insertion.

The Root Node

First, let's go over the root node.

```
class Forest():
    def __init__(self):
        self.root = CommentNode({ "id": "abc123" })
```

The only key in the dictionary passed into `CommentNode` is `id`, therefore the root `CommentNode` will only contain the attributes `self.id` and `self.replies`. A mock submission ID is shown. The actual source code will pull the submission's ID based on the URL that was passed into the `-c` flag and set the `id` value accordingly.

Before I get to the insertion methods, I will explain how comments and their replies are linked.

How PRAW Comments Are Linked

`PRAW` returns all submission comments by level order. This means all top levels are returned first, followed by all second-level replies, then third, so on and so forth.

I will create some mock comment objects to demonstrate. Here is a top level comment corresponding to the mock submission ID. Note the `parent_id` contains the submission's `id`, which is stored in `self.root.id`:

```
{
  "author": "u/asdfasdfasdf",
  "body": "A top level comment here.",
  "created_utc": "06-06-2006 06:06:06",
  "distinguished": null,
  "edited": false,
  "id": "qwerty1",
  "is_submitter": false,
  "link_id": "t3_asdfgh",
  "parent_id": "t3_abc123",
  "score": 666,
  "stickied": false
}
```

Here is a second-level reply to the top comment. Note the `parent_id` contains the top comment's `id`:

```
{
  "author": "u/hjklhjklhjklhjkl",
  "body": "A reply here.",
  "created_utc": "06-06-2006 18:06:06",
  "distinguished": null,
  "edited": false,
  "id": "hjkl234",
  "is_submitter": true,
  "link_id": "t3_1a2b3c",
  "parent_id": "t1_qwerty1",
  "score": 6,
  "stickied": false
}
```

This pattern continues all the way down to the last level of comments. It is now very easy to link the correct comments together. I do this by calling `split("_", 1)` on the `parent_id` and then getting the second item in the split list to compare values. I also specify the `maxsplit` parameter to force one split.

The Insertion Methods

I then defined the methods for `CommentNode` insertion.

```

def _dfs_insert(self, new_comment):
    stack = []
    stack.append(self.root)

    visited = set()
    visited.add(self.root)

    found = False
    while not found:
        current_comment = stack.pop(0)

        for reply in current_comment.replies:
            if new_comment.parent_id.split("_", 1)[1] == reply.id:
                reply.replies.append(new_comment)
                found = True
            else:
                if reply not in visited:
                    stack.insert(0, reply)
                    visited.add(reply)

def seed(self, new_comment):
    parent_id = new_comment.parent_id.split("_", 1)[1]

    self.root.replies.append(new_comment) \
        if parent_id == getattr(self.root, "id") \
        else self._dfs_insert(new_comment)

```

I implemented the [depth-first search](#) algorithm to find a comment's parent node and insert it into the parent node's `replies` array. I defined a separate `visited` set to keep track of visited `CommentNode`s to avoid an infinite loop of inserting `CommentNode`s that were already visited into the `stack`. At first I wrote a recursive version of depth-first search, but then opted for an iterative version because it would not scale well for submissions that included large amounts of comments, ie. stack overflow.

Within the `seed` method, I first check if the `CommentNode` is a top level comment by comparing its parent ID to the submission ID. Depth-first search is triggered if the `CommentNode` is not a top level comment.

Serializing the Forest

Since Python's built-in JSON module can only handle primitive types that have a direct JSON equivalent, a custom encoder is necessary to convert the `Forest` into JSON format. I defined this in `Export.py`.

```

from json import JSONEncoder

class EncodeNode(JSONEncoder):
    def default(self, object):
        return object.__dict__

```

The `default()` method overrides `JSONEncoder`'s `default()` method and serializes the `CommentNode` by converting it into a dictionary, which is a primitive type that has a direct JSON equivalent:

```
EncodeNode().encode(CommentNode)
```

This ensures the node is correctly encoded before I call the `seed()` method to insert a new `CommentNode` into the `replies` arrays of its respective parent `CommentNode`.

I can then use this custom `JSONEncoder` subclass while exporting by specifying it within `json.dump()` with the `cls` kwarg:

```
with open(filename, "w", encoding = "utf-8") as results:  
    json.dump(data, results, indent = 4, cls = EncodeNode)
```

This was how the structured comments export was implemented. Refer to the source code located in `urs/praw_scrapers/Comments.py` to see more. I hope this was somewhat interesting and/or informative. Thanks for reading!

Speeding Up Python with Rust

Created: May 05, 2023

Table of Contents

- [Introduction](#)
- [Some Reasons Why I Love Rust](#)
 - [Great Performance Without Sacrificing Memory Safety](#)
 - [All Bases Covered](#)
 - [Compiler Error Messages Are Actually Useful](#)
 - [Fantastic Developer Ecosystem](#)
- [Controllable Bottlenecks in URS](#)
- [Depth-First Search](#)
- [Submission Comments Returned by PRAW](#)
- [Translating Python to Rust](#)
- [The Performance Increase](#)
 - [Rust vs. Python Demo](#)

Introduction

I was only proficient in Python when I initially created `URS` in 2019. Four years later, I have accumulated experience with a variety of programming languages such as JavaScript, Java, Go, and Rust. Of all the languages I have tried, I have found Rust was the only one that does everything better than all other programming languages, learning curve notwithstanding.

This document briefly details how I wrote a Python module in Rust to improve the performance of the submission comments scraper, particularly when creating the structured comments JSON format.

Some Reasons Why I Love Rust

I will try to keep this section short, but here are a few major reasons why I love Rust. Feel free to skip this section.

Great Performance Without Sacrificing Memory Safety

The performance you get from Rust programs are as fast as, or even faster, than those written in C or C++. The difference is Rust offers a unique way for developers to manage

memory and makes it less likely for a program to have memory leaks. You have to make a conscious decision to make your Rust program *not* memory safe. On the other hand, it is extremely easy to accidentally mess up memory management in C or C++, causing all kinds of bad shit to happen.

All Bases Covered

Assuming you have followed good Rust development practices, you do not need to risk running into a runtime error when you deploy your application because you have already handled what should happen if something within your application fails. This means there is (typically) no unexpected behavior, which means it is less likely to crash and, if used in a professional setting, you are less likely to be called into work to fix something during the weekend just because something crashed.

Getting used to this paradigm has had a net positive on the way I write code, especially in dynamically interpreted programming languages such as Python and JavaScript. I have become more aware of where runtime errors may potentially occur and try my best to handle them in advance.


Compiler Error Messages Are Actually Useful

The title says it all -- the error messages you see when you get compiler errors are actually useful and will clearly tell you exactly where the error is, why it happened, and what you need to do to fix it. None of that lengthy, useless bullshit you might see from an error raised in a JavaScript or Java program, for example.

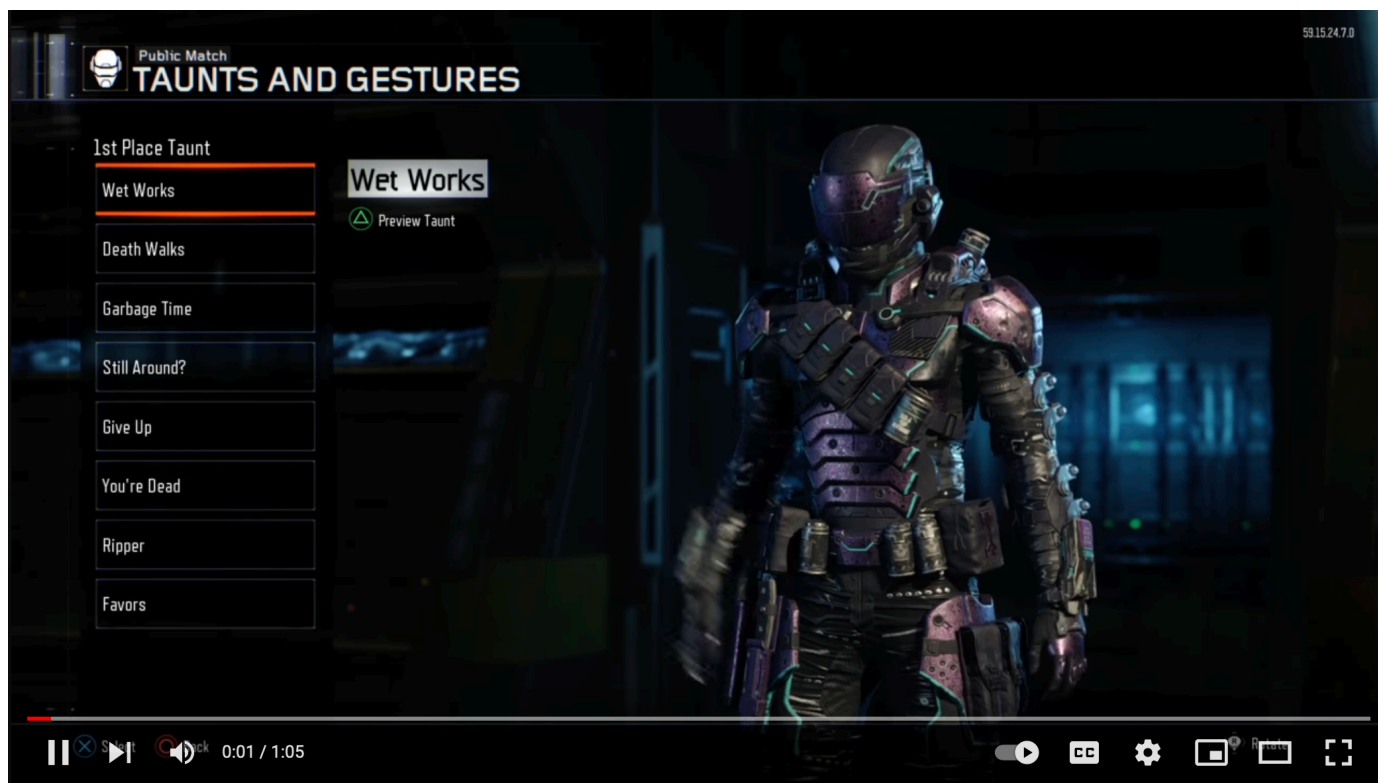
Here is an example of an error message you might see when compiling a Rust program:

useful Rust compiler error message

Now compare it to this:

JavaScript is fucking garbage

All I can say (on Rust's behalf) is:



Click the screenshot to play.

Fantastic Developer Ecosystem

`cargo` is an amazing CLI tool, making it incredibly easy to manage Rust projects. Here are some of the most notable features:

- Managing (adding/modifying/removing) dependencies
- Compiling your project
- Formatting and linting code
- Generating documentation from the docstrings you have written within your code
- Publishing your project to crates.io and documentation to docs.rs

Additionally, the `Cargo.toml` file makes it very easy to set project metadata. Thankfully Python's `Poetry` brings a `cargo`-esque experience to Python projects, making the old `setup.py` + `requirements.txt` project format feel very outdated (which it is, as `pyproject.toml` is now the new standard way to specify project metadata per [PEP 621](https://pep.python.org/pep-0621/)).

Upgrading Rust is also a very simple `rustup upgrade` command in the terminal.

I will leave the list at that for now, but I strongly encourage anyone who is looking to learn something new to [learn Rust](https://www.rust-lang.org/learn).

Controllable Bottlenecks in URS

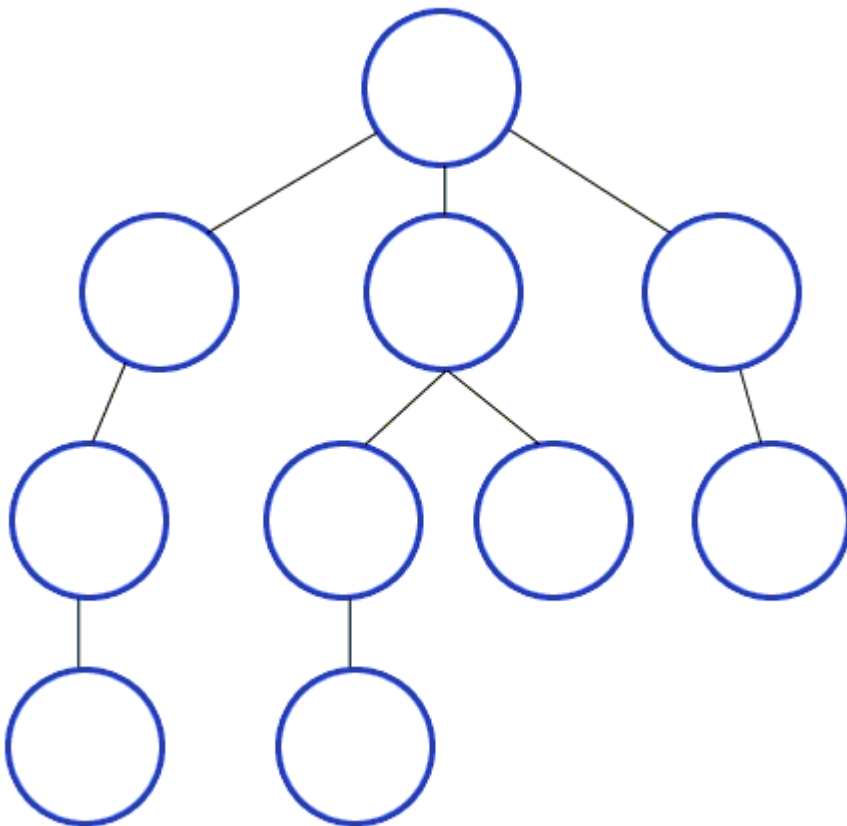
Diving into the practical reasons as to why I rewrote parts of **URS** in Rust, we first have to consider performance bottlenecks that I can optimize.

The biggest bottlenecks for how fast **URS** can scrape Reddit are the user's internet connection speed and the number of results returned when scraping Subreddits, Redditors, or submission comments. Unfortunately those variables are out of my control, so there is nothing I can do to optimize on that end. However, I *can* optimize the speed of which the raw data returned from the Reddit API is processed.

The depth-first search algorithm that is used when exporting structured comments is the most obvious performance bottleneck, which is why I rewrote this functionality in Rust.

Depth-First Search

Here's a quick refresher for the algorithm. Depth-first search is a search algorithm that traverses depth-first within a tree to find the target node, traversing deep into a node before moving on to the next in the tree. Here is a GIF visualizing the algorithm's steps:



The time complexity for this algorithm is $O(V + E)$, where V is the number of vertices and E is the number of edges in the tree, since the algorithm explores each vertex and edge exactly one time.

Submission Comments Returned by PRAW

Submission comments are returned by PRAW in order level. This means we get a **PHAT** list of comments containing all top-level comments first, followed by second-level comments, third-level comments, so on and so forth. Refer to the ["How PRAW Comments Are Linked" section in The Forest](#) to see an example of what I am describing.

The more comments are processed while creating the structured JSON via the depth-first search algorithm, the more deeply nested those comments are within the existing comment thread structure.

It would be ideal to have this algorithm run as fast as possible, especially for Reddit submissions that contain a substantial number of comments, such as some r/AskReddit threads.

Translating Python to Rust

I was able to write a Python module in Rust via the `PyO3` crate, using `maturin` to compile my Rust code and install it into my Python virtual environment. Refer to the linked documentation to learn more about how you can integrate Rust with your own Python project.

I will not explain every single line of the Python to Rust translation since a Rust/ `PyO3` / `maturin` tutorial is not within the scope of this document, but I will paste the depth-first search algorithm to showcase the nearly direct 1-to-1 translation between the two languages.

Here is the code for the original Python implementation:

```
class Forest():
    def _dfs_insert(self, new_comment: CommentNode) -> None:
        stack = []
        stack.append(self.root)

        visited = set()
        visited.add(self.root)

        found = False
        while not found:
            current_comment = stack.pop(0)

            for reply in current_comment.replies:
                if new_comment.parent_id.split("_", 1)[1] == reply.id:
                    reply.replies.append(new_comment)
                    found = True
                else:
                    if reply not in visited:
                        stack.insert(0, reply)
                        visited.add(reply)

    def seed(self, new_comment: CommentNode) -> None:
        parent_id = new_comment.parent_id.split("_", 1)[1]

        if parent_id == getattr(self.root, "id"):
            self.root.replies.append(new_comment)
        else:
            self._dfs_insert(new_comment)
```

Here is the Rust translation:

```

impl Forest {
    fn _dfs_insert(&mut self, new_comment: CommentNode) {
        let root_id = &self.root.id.clone();

        let mut stack: VecDeque<&mut CommentNode> = VecDeque::new();
        stack.push_front(&mut self.root);

        let mut visited: HashSet<String> = HashSet::new();
        visited.insert(root_id.to_string());

        let target_id = &new_comment
            .parent_id
            .split('_')
            .last()
            .unwrap_or(&new_comment.parent_id)
            .to_string();

        let mut found = false;

        while !found {
            if let Some(comment_node) = stack.pop_front() {
                for reply in comment_node.replies.iter_mut() {
                    if target_id == &reply.id {
                        reply.replies.push(new_comment.clone());
                        found = true;
                    } else {
                        let child_id = reply.id.clone();

                        if !visited.contains(child_id.as_str()) {
                            stack.push_front(reply);
                            visited.insert(child_id);
                        }
                    }
                }
            }
        }
    }

    fn seed_comment(&mut self, new_comment: CommentNode) {
        let parent_id = &new_comment
            .parent_id
            .split('_')
            .last()
            .unwrap_or(&new_comment.parent_id)
            .to_string();

        if parent_id == &self.root.id {
            self.root.replies.push(new_comment);
        } else {
            self._dfs_insert(new_comment);
        }
    }
}

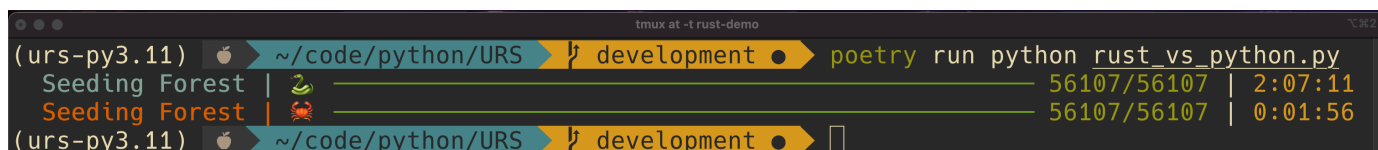
```

Refer to the [taisun/](#) directory to view the full Rust implementation.

The Performance Increase

I have no words for how much faster the Rust implementation is compared to the original Python implementation. All I can do is provide you with this screenshot and let you interpret the results for yourself. The timestamp in the far right column is the time elapsed in

HH:MM:SS.



```
(urs-py3.11) ~/code/python/URS development poetry run python rust_vs_python.py
Seeding Forest | 🐍 56107/56107 | 2:07:11
Seeding Forest | 🦀 56107/56107 | 0:01:56
(urs-py3.11) ~/code/python/URS development
```

I hope the emojis make it obvious which progress bar corresponds to which implementation, but just to be extra clear: 🐍 = Python, 🦀 = Rust.

This test was run on a 2021 MacBook Pro with the binned M1 Pro chip.

Rust vs. Python Demo

You do not have to take my word for it -- I have put together a small demo located on the [rust-demo](#) branch in the repository.

The demo branch includes a JSON file which contains 56,107 unstructured comments scraped from [this r/AskReddit post](#). The demo Python script will read the unstructured comments from the JSON file and run the depth-first search algorithm in both Python and Rust concurrently. Progress bars for each task will visualize the progress made with each implementation and display the time elapsed at the far right column.

Run the following commands to install and run the demo:

```
git clone -b rust-demo https://github.com/JosephLai241/URS.git --depth=1
poetry install
```

```
# You do not need to run the following command if the virtualenv is already
# activated.
```

```
#
```

```
# If this command fails/does not activate the virtualenv, run
```

```
# `source .venv/bin/activate` to activate the virtualenv created by `Poetry`.
poetry shell
```

```
maturin develop --release
```

```
poetry run python rust_vs_python.py
```

Before Making Pull or Feature Requests

Consider the scope of this project before submitting a pull or feature request. **URS** stands for Universal Reddit Scraper. Two important aspects are listed in its name - *universal* and *scraper*.

I will not approve feature or pull requests that deviate from its sole purpose. This may include scraping a specific aspect of Reddit or [adding functionality that allows you to post a comment with URS](#). Adding either of these requests will no longer allow **URS** to be universal or merely a scraper. However, I am more than happy to approve requests that enhance the current scraping capabilities of **URS**.

Building on Top of URS

Although I will not approve requests that deviate from the project scope, feel free to reach out if you have built something on top of **URS** or have made modifications to scrape something specific on Reddit. I will add your project to the [Derivative Projects](#) section!

Making Pull or Feature Requests

You can suggest new features or changes by going to the [Issues tab](#) and fill out the Feature Request template. If there is a good reason for a new feature, I will consider adding it.

You are also more than welcome to create a pull request -- adding additional features, improving runtime, or refactoring existing code. If it is approved, I will merge the pull request into the master branch and credit you for contributing to this project.

Contributors

Date	User	Contribution
March 11, 2020	ThereGoesMySanity	Created a pull request adding 2FA information to README
October 6, 2020	LukeDSchenk	Created a pull request fixing "[Errno 36] File name too long" issue, making it impossible to save comment scrapes with long titles
October 10, 2020	IceBerge421	Created a pull request fixing a cloning error occurring on Windows machines due to illegal file name characters, " <code>"</code> , found in two scrape samples

Derivative Projects

This is a showcase for projects that are built on top of URS!

skiwheelr/URS

```
Mentions of DD in r/WSB: 4466 & in links: 120337
Mentions of GME in r/WSB: 1461 & in links: 33270
Mentions of BB in r/WSB: 588 & in links: 14684
Mentions of APPL in r/WSB: 73 & in links: 3574
Mentions of ETH in r/WSB: 382 & in links: 10402
Mentions of AMC in r/WSB: 604 & in links: 7599
Mentions of BABA in r/WSB: 17 & in links: 328
Mentions of BBBY in r/WSB: 30 & in links: 365
Mentions of BTC in r/WSB: 0 & in links: 102
Mentions of AMD in r/WSB: 36 & in links: 787
Mentions of AZN in r/WSB: 8 & in links: 195
Mentions of ADMS in r/WSB: 0 & in links: 3
Mentions of AMRS in r/WSB: 0 & in links: 2
Mentions of IFF in r/WSB: 141 & in links: 3027
Mentions of NOK in r/WSB: 451 & in links: 3624
Mentions of DOGE in r/WSB: 1 & in links: 111
Mentions of SNDL in r/WSB: 5 & in links: 45
Mentions of NIKE in r/WSB: 1 & in links: 24
```

Contains a bash script built on URS which counts ticker mentions in Subreddits, subsequently cURLs all the relevant links in parallel, and counts the mentions of those.