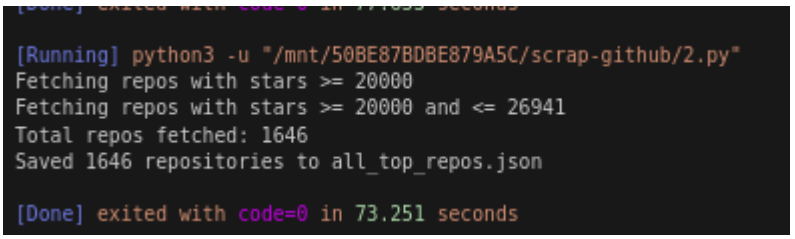
1. Initially, I directly made a call to the GitHub REST API to retrieve the results for all repositories with stars greater than 20,000, but GitHub has a cap of 1000 on the maximum number of repositories it can return for a given filter. Hence, I didn't get results directly; I had to split the queries into different sections.

2. Therefore, I decided to split the queries into sections. The first query finds the top 1000 results, then the next range query finds the lowest no of stars in the previous section's query response, and based on that the query is updated to find all repos with no. of stars >=20,000 and <= to lowest no. of stars of the previous query and so on. Moreover, the response is also deduplicated to remove multiple occurrences of a single repo.

3. It might be possible that we have multiple repos with the same number of stars(say 26,000). Then, since in the query filter we will only receive 1,000 repos, It might be possible that some repos with stars = 26,000 will be included in those 1,000 repos, but rest don't. Therefore, even in the next section of the queries, I have a max limit as repo with stars = previous_repo_lowest_stars(say 26,000).

4. We had deduplicated because the next section has repo >=previous_repo_lowest_stars, Therefore, the next response will include repo with star= previous_repo_lowest_stars, but some of those repos might be already included in the previous section response.

5. It took about 70 seconds to fetch all the repos by executing the 1.py file, and then around 1800 seconds(~30 minutes) to fetch the website corresponding to each repo by executing the 2.py file. The higher time in 2.py file execution is due to the rate limit set by GitHub.



6. The response from the fetch request to "api.github.com/search/repositories" doesn't provide us with the URL of the website of the owner. Although it has a key called "homepage", when I manually looked at those links then those links were not present in most cases, and in some cases, the links present were leading specific sections of their website-related donation, etc.. instead of homepage). Therefore, we have to make separate API calls to "https://api.github.com/orgs/{owner_login}" or

"https://api.github.com/users/{owner_login}" depending upon whether the owner of the repo is a user or an organization (owner type is available in the JSON response of the 1.py file).

7. Now, since we have made separate API calls for each repo, GitHub's API request rate limit doesn't allow us to fetch the results for all the repos together in one go, and we have to add delays in between fetch requests. Hence, I decided to have a separate file named 2.py to fetch those website links. And due to those delays and one-by-one requests, the time to fetch website links for all repos went close to 30 minutes.

8. Once we got the website links for all the repos, I saved them in JSON format. For each response, we have 4 fields corresponding to repo_full_name, owner, owner_type, and website.

Later, from that JSON file, I got my “final_output.xlsx” file. First few rows of the “final_output.xlsx” file:

	A	B	C	
1	repo_full_name	owner	owner_type	website
2	freeCodeCamp/freeCodeCamp	freeCodeCamp	Organization	https://www.freecodecamp.org
3	codecrafters-io/build-your-own-x	codecrafters-io	Organization	https://codecrafters.io
4	sindresorhus/awesome	sindresorhus	User	https://sindresorhus.com/apps
5	EbookFoundation/free-programming-books	EbookFoundation	Organization	https://ebookfoundation.org/
6	public-apis/public-apis	public-apis	Organization	https://github.com/public-apis
7	kamranahmedse/developer-roadmap	kamranahmedse	User	https://kamranahmed.info
8	jwasham/coding-interview-university	jwasham	User	https://startupnextdoor.com
9	donnemartin/system-design-primer	donnemartin	User	http://donnemartin.com/