
```
!pip install jovian --upgrade --quiet
import jovian
jovian.commit(project="scraping-top-imdb-animated-series")
```

```
[jovian] Updating notebook "mdfaizan5262426/scraping-top-imdb-animated-series" on https://jovian.com
[jovian] Committed successfully! https://jovian.com/mdfaizan5262426/scraping-top-imdb-animated-series
```

```
'https://jovian.com/mdfaizan5262426/scraping-top-imdb-animated-series'
```

Scraping Top Animated Series from IMDB

- **Web scraping** is an automatic method to obtain large amounts of data from websites. Most of this data is unstructured data in an HTML format which is then converted into structured data in a spreadsheet or a database so that it can be used in various applications.

- The Internet Movie Database (IMDb) is an online database containing information and statistics about movies, TV shows and video games as well as actors, directors and other film industry professionals.
- The tools we are using are Python, requests, Beautiful Soup, Pandas

Outline of the project:

- We're going to scrape https://www.imdb.com/search/title/?at=0&genres=animation&keywords=anime&num_votes=1000,&sort=user_rating&title_type=tv_series
- We'll get a list of animated series.First for each series we'll take series title, rating, url and description.
- From the list of animated series, we retrive the top 100 series based on rating.
- We'll scrape all 100 series individually using it's urls and extract year duration, creator name, awards and nominations.
- At last we'll create a DataFrame in which all the extracted information will be displayed.
- Then we'll create CSV file in the following format:
- From top 100 animated series we'll save the data into a single CSV files.

  title, **rating**, **description**,**year_duration**, **creator_name**, **awards_and_nominations**, **url**

  Bleach:Thousand-Year Blood War, 9.4, The peace is suddenly broken when warning sirens blare through the Soul Society. Residents are disappearing without a trace and nobody knows who's behind it. Meanwhile, a darkness is approaching Ichigo and his friends in Karakura Town, 2022– , Tite Kubo, NaN, https://www.imdb.com//title/tt14986406/?ref_=adv_li_tt.

Scrape the list of top 100 anime based on user rating from IMDB

- Use rerequests to download the page
- Use BS4 to parse and extract information
- Convert to a pandas dataframe

```
!pip install requests --upgrade --quiet
!pip install beautifulsoup4 --upgrade --quiet
!pip install pandas --quiet

import requests
from bs4 import BeautifulSoup
import pandas as pd

def get_anime_page():
    anime_url = 'https://www.imdb.com/search/title/?at=0&genres=animation&keywords=anim
    response = requests.get(anime_url)
    print(response)
    if response.status_code != 200:
        raise Exception ('Failed to load page {} '.format(anime_url))
    doc = BeautifulSoup(response.text, 'html.parser')
    return doc
```

First we'll install and import required libraries.

`get_anime_page` function takes `anime_url` as input url and returns doc, using requests we'll check the status code of the page.

`Status_code` 200 means the page has successfully responded to our request, if not it will raise an exception.

BeautifulSoup takes the response and with **html.parser** it parses the page and all the information is stored in html format in doc.

```
doc = get_anime_page()
```

```
<Response [200]>
```

As you can see `get_anime_page` returns in html format, we've taken **head** element

```
doc.find('head')
```

```
<head>
<meta charset="utf-8"/>
<script type="text/javascript">var IMDbTimer={starttime: new
Date().getTime(),pt:'java'};</script>
<script>
    if (typeof uet == 'function') {
      uet("bb", "LoadTitle", {wb: 1});
    }
</script>
<script>(function(t){ (t.events = t.events || {})["csm_head_pre_title"] = new
Date().getTime(); })(IMDbTimer);</script>
<title>TV Series,
Rating Count at least 1,000,
Animation,
anime
(Sorted by IMDb Rating Descending) - IMDb</title>
<script>(function(t){ (t.events = t.events || {})["csm_head_post_title"] = new
Date().getTime(); })(IMDbTimer);</script>
<script>
    if (typeof uet == 'function') {
      uet("be", "LoadTitle", {wb: 1});
    }
</script>
<script>
    if (typeof uex == 'function') {
      uex("ld", "LoadTitle", {wb: 1});
    }
</script>
<link href="https://www.imdb.com/search/title/?
```

```
title_type=tv_series&amp;genres=animation&amp;keywords=anime" rel="canonical"/>
<meta content="http://www.imdb.com/search/title/?
title_type=tv_series&amp;genres=animation&amp;keywords=anime" property="og:url">
<script>
    if (typeof uet == 'function') {
      uet("bb", "LoadIcons", {wb: 1});
    }
</script>
<script>(function(t){ (t.events = t.events || {})["csm_head_pre_icon"] = new
Date().getTime(); })(IMDbTimer);</script>
<link href="https://m.media-amazon.com/images/G/01/imdb/images-
ANDW73HA/favicon_desktop_32x32._CB1582158068_.png" rel="icon" sizes="32x32"/>
<link href="https://m.media-amazon.com/images/G/01/imdb/images-
ANDW73HA/favicon_iPad_retina_167x167._CB1582158068_.png" rel="icon" sizes="167x167"/>
<link href="https://m.media-amazon.com/images/G/01/imdb/images-
ANDW73HA/favicon_iPhone_retina_180x180._CB1582158069_.png" rel="icon" sizes="180x180"/>
<link href="https://m.media-amazon.com/images/G/01/imdb/images-ANDW73HA/apple-touch-
icon-mobile._CB479963088_.png" rel="apple-touch-icon"/>
<link href="https://m.media-amazon.com/images/G/01/imdb/images-ANDW73HA/apple-touch-
icon-mobile-76x76._CB479962152_.png" rel="apple-touch-icon" sizes="76x76"/>
<link href="https://m.media-amazon.com/images/G/01/imdb/images-ANDW73HA/apple-touch-
icon-mobile-120x120._CB479963088_.png" rel="apple-touch-icon" sizes="120x120"/>
<link href="https://m.media-amazon.com/images/G/01/imdb/images-ANDW73HA/apple-touch-
icon-web-152x152._CB479963088_.png" rel="apple-touch-icon" sizes="152x152"/>
<link href="https://m.media-amazon.com/images/G/01/imdb/images-ANDW73HA/android-mobile-
196x196._CB479962153_.png" rel="shortcut icon" sizes="196x196"/>
<meta content="#000000" name="theme-color"/>
<link href="https://m.media-amazon.com/images/S/sash/MzfIBMq9GBucYqW.xml" rel="search"
title="IMDb" type="application/opensearchdescription+xml"/>
<script>(function(t){ (t.events = t.events || {})["csm_head_post_icon"] = new
Date().getTime(); })(IMDbTimer);</script>
<script>
    if (typeof uet == 'function') {
      uet("be", "LoadIcons", {wb: 1});
    }
</script>
<script>
    if (typeof uex == 'function') {
      uex("ld", "LoadIcons", {wb: 1});
    }
</script>
<meta content="advsearch" property="pageType"/>
<meta content="title" property="subpageType"/>
<link href="https://m.media-
amazon.com/images/G/01/imdb/images/social/imdb_logo._CB410901634_.png"
rel="image_src"/>
<meta content="https://m.media-
amazon.com/images/G/01/imdb/images/social/imdb_logo._CB410901634_.png"
property="og:image"/>
```

```
<meta content="115109575169727" property="fb:app_id"/>
<meta content="TV Series,
Rating Count at least 1,000,
Animation,
anime
(Sorted by IMDb Rating Descending) - IMDb" property="og:title"/>
<meta content="IMDb" property="og:site_name"/>
<meta content="TV Series,
Rating Count at least 1,000,
Animation,
anime
(Sorted by IMDb Rating Descending) - IMDb" name="title"/>
<meta content="IMDb's advanced search allows you to run extremely powerful queries over
all people and titles in the database. Find exactly what you're looking for!"
name="description"/>
<meta content="IMDb's advanced search allows you to run extremely powerful queries over
all people and titles in the database. Find exactly what you're looking for!"
property="og:description"/>
<meta content="EVNNZNVB626194Y4A7TP" name="request_id"/>
<script>
        (function (win) {
            win.PLAID_LOAD_FONTS_FIRED = true;

            if (typeof win.FontFace !== "undefined"
                && typeof win.Promise !== "undefined") {
                if (win.ue) {
                    win.uet("bb", "LoadRoboto", { wb: 1 });
                }
                var allowableLoadTime = 1000;
                var startTimeInt = +new Date();
                var roboto = new FontFace('Roboto',
                    'url(https://m.media-amazon.com/images/G/01/IMDb/cm9ib3Rv.woff2)',
                    { style:'normal', weight: 400 });
                var robotoMedium = new FontFace('Roboto',
                    'url(https://m.media-
amazon.com/images/G/01/IMDb/cm9ib3RvTWVk.woff2)',
                    { style:'normal', weight: 500 });
                var robotoBold = new FontFace('Roboto',
                    'url(https://m.media-
amazon.com/images/G/01/IMDb/cm9ib3RvQm9sZA.woff2)',
                    { style:'normal', weight: 600 });
                var robotoLoaded = roboto.load();
                var robotoMediumLoaded = robotoMedium.load();
                var robotoBoldLoaded = robotoBold.load();

                win.Promise.all([robotoLoaded, robotoMediumLoaded,
robotoBoldLoaded]).then(function() {
                    var loadTimeInt = +new Date();
                    var robotoLoadedCount = 0;
```

```
                    if ((loadTimeInt - startTimeInt) <= allowableLoadTime) {
                        win.document.fonts.add(roboto);
                        win.document.fonts.add(robotoMedium);
                        win.document.fonts.add(robotoBold);
                        robotoLoadedCount++;
                    }
                    if (win.ue) {
                        win.ue.count("roboto-loaded", robotoLoadedCount);
                        win.uet("be", "LoadRoboto", { wb: 1 });
                        win.uex("ld", "LoadRoboto", { wb: 1 });
                    }
                }).catch(function() {
                    if (win.ue) {
                        win.ue.count("roboto-loaded", 0);
                    }
                });
            } else {
                if (win.ue) {
                    win.ue.count("roboto-load-not-attempted", 1);
                }
            }
        })(window);
    </script>
<script>
    if (typeof uet == 'function') {
      uet("bb", "LoadCSS", {wb: 1});
    }
</script>
<script>(function(t){ (t.events = t.events || {})["csm_head_pre_css"] = new
Date().getTime(); })(IMDbTimer);</script>
<link href="https://m.media-amazon.com/images/S/sash/pyjhBimOKeDFYm5.css"
rel="stylesheet" type="text/css">
<!-- h=ics-c52xl-13-1f-0c3e6452.us-east-1 -->
<!--[if IE]><link rel="stylesheet" type="text/css" href="https://m.media-
amazon.com/images/S/sash/pXHSPBTKPo0GIjW.css" /><![endif]-->
<link href="https://m.media-amazon.com/images/S/sash/vEv7EhTS45PyD8e.css"
rel="stylesheet" type="text/css"/>
<link href="https://m.media-amazon.com/images/S/sash/i-Y65BmJrKWIfbK.css"
rel="stylesheet" type="text/css"/>
<link href="https://m.media-amazon.com/images/S/sash/HqcrRlviNTScutU.css"
rel="stylesheet" type="text/css"/>
<link href="https://m.media-amazon.com/images/S/sash/mrwUCCyp3V14GU0.css"
rel="stylesheet" type="text/css"/>
<link href="https://m.media-amazon.com/images/S/sash/sHfJ8wTsDnFAHcY.css"
rel="stylesheet" type="text/css"/>
<noscript>
<link href="https://m.media-amazon.com/images/S/sash/CCc6Ja$8QUPPKkY.css"
rel="stylesheet" type="text/css"/>
</noscript>
```

```
<script>(function(t){ (t.events = t.events || {})["csm_head_post_css"] = new
Date().getTime(); })(IMDbTimer);</script>
<script>
    if (typeof uet == 'function') {
      uet("be", "LoadCSS", {wb: 1});
    }
</script>
<script>
    if (typeof uex == 'function') {
      uex("ld", "LoadCSS", {wb: 1});
    }
</script>
<script>
    if (typeof uet == 'function') {
      uet("bb", "LoadJS", {wb: 1});
    }
</script>
<script>
    if (typeof uet == 'function') {
      uet("bb", "LoadHeaderJS", {wb: 1});
    }
</script>
<script>(function(t){ (t.events = t.events || {})["csm_head_pre_ads"] = new
Date().getTime(); })(IMDbTimer);</script>
<script type="text/javascript">
            // ensures js doesn't die if ads service fails.
            // Note that we need to define the js here, since ad js is being rendered
inline after this.
            (function(f) {
                // Fallback javascript, when the ad Service call fails.

                if((window.csm == null || window.generic == null || window.consoleLog
== null)) {
                    if (window.console && console.log) {
                        console.log("one or more of window.csm, window.generic or
window.consoleLog has been stubbed...");
                    }
                }

                window.csm = window.csm || { measure:f, record:f, duration:f, listen:f,
metrics:{} };
                window.generic = window.generic || { monitoring: { start_timing: f,
stop_timing: f } };
                window.consoleLog = window.consoleLog || f;
            })(function() {});
        </script>
<script type="text/javascript">window.useRatingTaskCompletion = true;</script>
<script>
    if ('csm' in window) {
```

```
      csm.measure('csm_head_delivery_finished');
    }
  </script>
<script>
    if (typeof uet == 'function') {
      uet("be", "LoadHeaderJS", {wb: 1});
    }
</script>
<script>
    if (typeof uex == 'function') {
      uex("ld", "LoadHeaderJS", {wb: 1});
    }
</script>
<script>
    if (typeof uet == 'function') {
      uet("be", "LoadJS", {wb: 1});
    }
</script>
<script>
    if (typeof uex == 'function') {
      uex("ld", "LoadJS", {wb: 1});
    }
</script>
</link></meta></head>
```

First we'll create a function called `get_anime_data1` , which will be used in the below functions.

As we're scrapping top 100 animated series from IMDb the first page of the website has only top 50 series but we need next 50 too, that's why we've taken two urls in imdb_url, first url contains top 50 and the second top 51-100.

The headers contains the **User-Agent** which is a string of information that identifies a user's browser and operating system.There will be certain limit for sending requests to website, after that the website will send 404 error to evade this we're using **headers** while sending request to website.

```
imdb_url = ['https://www.imdb.com/search/title/?at=0&genres=animation&keywords=anime&nu
headers = {"User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (

def get_anime_data1(Urls):
    anime_data1 = []

    for i in Urls:
        response = requests.get(i,headers = headers)
        doc = BeautifulSoup(response.content, 'html.parser')
        anime_data1.append(doc)
    return anime_data1
```

`get_anime_data1` takes one arguement which is urls and returns `anime_data1` .

We're using **for loop** to iterate over the url and parse it with **BeautifulSoup** depending on the urls the `anime_data1` will get appended. We're using only two urls below theerefore the for loop will iterate twice.

```
anime_data3 = get_anime_data1(imdb_url)
```

We're just calling the `get_anime_data1` function and the argument given is `imdb_url` which contains two **urls** for top 100 animated series.

Let's create some helper functions to parse information from the page.

To get **anime titles**, we can pick **div** tags and **class**:lister-item mode-advanced inside this we'll find **h3** tags which contains anime titles


```
def get_anime_titles():
    anime_title = []
    #The anime_data3 conatins all the webpage data in html format
    for i in anime_data3:
        #find_all is BeautifulSoup method used for finding specified tags in html
        anime_data = i.find_all('div', attrs={'class': 'lister-item mode-advanced'})
        # Get anime title using h3 tags
        for each_line in anime_data:
            title = each_line.h3.a.text
            anime_title.append(title)
    return anime_title
```

`get_anime_titles` can be used to get the list of titles.
using **find_all** method we'll get required tags furthermore looping over those tags we can find **h3** tags which

contains **titles** and using **.text** method we extract only text from tags.

```
titles = get_anime_titles()
```

Below is the **titles** for top 10 animated series

```
titles[:10]
```

```
['Fullmetal Alchemist: Brotherhood',
 'Attack on Titan',
 'Hunter x Hunter',
 'One Piece',
 'Cowboy Bebop',
 'Vinland Saga',
 'Dragon Ball Z',
 'Steins;Gate',
 'Demon Slayer: Kimetsu no Yaiba',
 'Naruto: Shippuden']
```

Similarly to get rating, we'll use same tag that is **div** tags but different **class** : inline-block ratings-imdb-rating.

```
def get_anime_rating():
    anime_rating = []
    #The anime_data3 conatins all the webpage data in html format
    for i in anime_data3:
        title_rating_tag = i.find_all('div',{'class':'inline-block ratings-imdb-rating'
        for j in title_rating_tag:
            rating_tag = j.text.replace('\n', '')
            anime_rating.append(rating_tag)
    return anime_rating
```

As explained above we'll use same process to extract rating, the **class** is different from before and the `rating_tag` contains **\n\n9.4\n** we need only rating that's why we're using **.replace** method to replace **\n** by **empty sapce**.

```
rating = get_anime_rating()
```

Rating for top 10 animated series

```
rating[:10]
```

```
['9.1', '9.0', '9.0', '8.9', '8.9', '8.8', '8.8', '8.8', '8.7', '8.7']
```

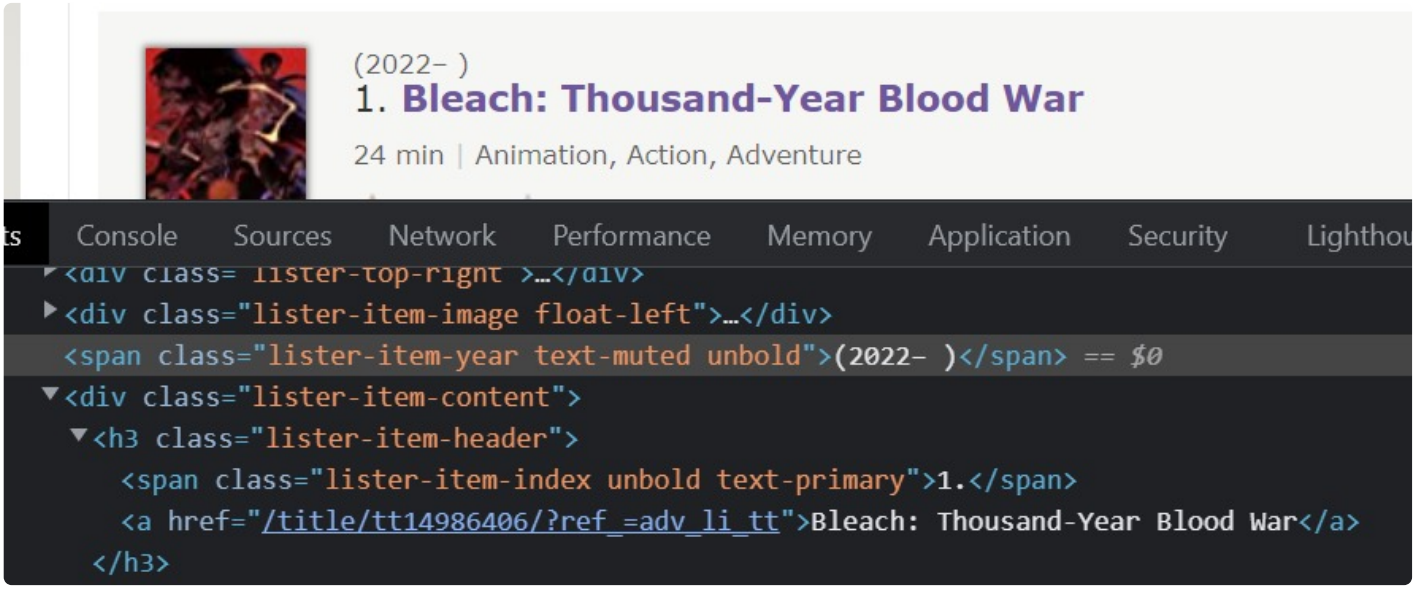
To get **url** we'll use **h3** tags and **class**:url.

```
def get_anime_url():
    anime_url_tags = []
    base_url = 'https://www.imdb.com/'
    #The anime_data3 conatins all the webpage data in html format
```

```
    for i in anime_data3:
        # Get url using h3 tags
        url_tags = i.find_all('h3', {'class':'lister-item-header'})

        for j in url_tags:
            # concatenate base_url and url obtained by h3 tags
            anime_url_tags.append('https://www.imdb.com/'+ j.find('a')['href'])
    return anime_url_tags
```

The `url_tags` contains **h3** tags which has **title**, **rank** and **url** but we require only **url** from `url_tags` therefore we'll use **for loop** in which the **find** method will get only **a href**.
But the **url** we got is incomplete therefore we'll use `base_url` to concatenate with the **url** we've obtained.




```
urls = get_anime_url()
```

url for the top 10 animated series

```
urls[:10]
```

```
['https://www.imdb.com//title/tt1355642/?ref_=adv_li_tt',
 'https://www.imdb.com//title/tt2560140/?ref_=adv_li_tt',
 'https://www.imdb.com//title/tt2098220/?ref_=adv_li_tt',
 'https://www.imdb.com//title/tt0388629/?ref_=adv_li_tt',
 'https://www.imdb.com//title/tt0213338/?ref_=adv_li_tt',
 'https://www.imdb.com//title/tt10233448/?ref_=adv_li_tt',
 'https://www.imdb.com//title/tt0214341/?ref_=adv_li_tt',
 'https://www.imdb.com//title/tt1910272/?ref_=adv_li_tt',
 'https://www.imdb.com//title/tt9335498/?ref_=adv_li_tt',
 'https://www.imdb.com//title/tt0988824/?ref_=adv_li_tt']
```

For **description** we'll use **p** tags and **class**: 'text-muted'

```
def get_anime_description():
    anime_description = []
    #The anime_data3 conatins all the webpage data in html format
```

```
    for i in anime_data3:
        # Get description using p tags
        anime_descp_tags = i.find_all('p', {'class': 'text-muted'})
        for j in anime_descp_tags:
            # Replace '\n' by empty space ''
            anime = j.text.replace('\n','')
            anime_description.append(anime)
        # Index the anime_descript list for only description
        anime_descript = anime_description[1::2]
    return anime_descript
```

`anime_descp_tags` contains description of an animated series but it has '/n' so we'll use **.replace** method to remove it.

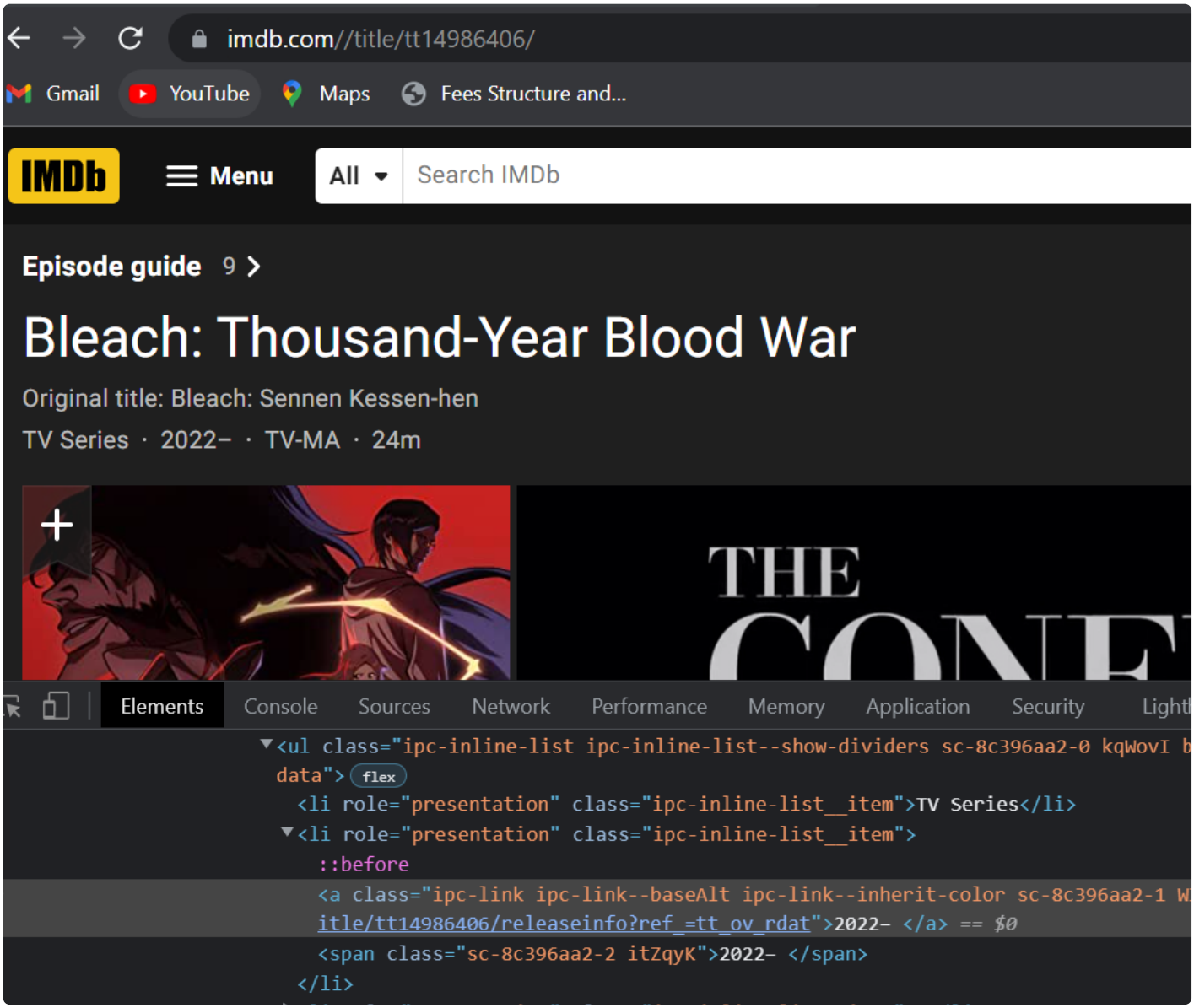
The `anime_description` contains the duration, genre and description of an animated series but we require only description therefore we'll store `anime_description` in another variable called `anime_descript` and we'll take only second element using slicing as shown above.

```
description = get_anime_description()
```

description for top 10 animated series

Get the individual anime data from list of top 100 anime

Now we'll scrape **top 100** animated series **individually** using their **url** to extract **year_duration**, **creator_name** and **awards_and_nominations**.



We're using the `urls` which we got from `get_anime_url`, it contains **urls** of **top 100** animated series,then we'll call `anime_data1` function and pass `urls` as argument and save it in `anime_data2`, at last we'll iterate over `anime_data2` using for loop and extract required information individually.

```
anime_data2 = get_anime_data1(urls)
```

First we'll extract **year_duration** using **span** tag and **class** : sc-8c396aa2-2 itZqyK.

```
def get_year_duration():
    year_durations = []
    #The anime_data2 conatins all the 100 individual webpage data in html format
    for i in anime_data2:
        year_duration_tag = i.find_all('span', {'class':'sc-8c396aa2-2 itZqyK'})[0]
        for year in year_duration_tag:
            year_durations.append(year.text)
    return  year_durations
```

`get_year_duration` funnction iterates over `anime_data2` using for loop, as explained earlier we'll parse using BeautifulSoup.

The for loop iterates over all the **100** urls in the `urls` and extracts anime year duration.

We need only **year duration** which is at zero index in the **span** tag.

```
year_duration = get_year_duration()
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipykernel_38/1491692037.py in <module>
----> 1 year_duration = get_year_duration()

/tmp/ipykernel_38/4107560098.py in get_year_duration()
      3     #The anime_data2 conatins all the 100 individual webpage data in html format
      4     for i in anime_data2:
----> 5         year_duration_tag = i.find_all('span', {'class':'sc-8c396aa2-2 itZqyK'})
[0]
      6         for year in year_duration_tag:
      7             year_durations.append(year.text)

IndexError: list index out of range
```

year duration for **top** 10 anime.

```
year_duration[:10]
```

Similarly to get **creator name** we need **a** tags, and **class** : 'ipc-metadata-list-item__list-content-item ipc-metadata-list-item__list-content-item--link'.

```
def get_creator_name():
    creators = []
    #The anime_data2 conatins all the 100 individual webpage data in html format
    for i in anime_data2:
        creator_name_tag = i.find_all('a', {'class':'ipc-metadata-list-item__list-conte
        for i in creator_name_tag:
            creators.append(i.text)
    return creators
```

```
creator_name = get_creator_name()
```

creator_name for **top 10** anime

```
creator_name[:10]
```

For **awards_and_nominations** we'll use **label** tags and **class**: ipc-metadata-list-item__list-content-item.

```
def get_awards_nominations():
    awards_nominations = []
    #The anime_data2 conatins all the 100 individual webpage data in html format
    for i in anime_data2:
```

```
        award_tags = i.find_all('label',{'class':'ipc-metadata-list-item__list-content-

        for i in award_tags:
            award = i.text
            if award[0] in '0987654321':
                awards_nominations.append(award)
            else:
                awards_nominations.append('NaN')
    return awards_nominations
```

Not every animated series will have awards and nominations , therefore we'll be inserting **NaN**, abbreviation of **Not a Number** to animated series which does not have any awards and nominations using **if-else**.

```
awards_and_nomination = get_awards_nominations()
```

Here are **awards_and_nominations** for **top 10** animated series.

```
awards_and_nomination[:10]
```

Let's put this all together into a single function

The function `scrape_individual_anime` which invokes above functions that we've created so far.

```
def scrape_individual_anime():

    anime_dict = {
        'title' : get_anime_titles(),
        'rating' : get_anime_rating(),
        'description' : get_anime_description(),
        'year_duration' :get_year_duration(),
        'creator_name'  :get_creator_name(),
        'awards_and_nominations' : get_awards_nominations(),
        'url' : get_anime_url()
    }
    return pd.DataFrame(anime_dict)
```

We'll use `anime_dict` dictionary which has **keys** and **values**, the keys will act as **column** and values as **rows** of a **DataFrame**.

```
scrape_individual_anime()
```

The above DataFrame contains **7 columns and 100 rows**, the 7 columns are **title** , **rating description**, **year_duration**, **creator_name**, **awards_and_nominations** and **url**

Saving the Dataframe into CSV file

`scrape_anime` will use the `scrape_individual_anime` function which returns the **DataFrame**, then we'll save it into a CSV file.

```
def scrape_anime():
    individual_df = scrape_individual_anime()
    individual_df.to_csv('Top100_Anime.csv', index=False)
```

By calling `scrape_anime` it just takes the dataframe and save it into CSV file named **Top100_Anime**.

```
scrape_anime()
```

As we can see the **CSV** file have been created successfully in the jupiter notebook homepage.

The **Top100_Anime CSV file** contains **title**, **rating**, **description**, **year duration**, **creator name**, **awards and nominations** and **url**.

Uploading the CSV file to **jovian** workspace

```
jovian.commit(files=["Top100_Anime.csv"])
```

Summary

- First we've imported required libraires and using BeautifullSoup we scrape the website for requried information.

- Then we create individual functions to get the data and in the end we have a mega function `scrape_individual_anime` that combines all the function defined earlier.
- At last the `scrape_anime` we'll call the mega function `scrape_individual_anime` and save the data into individual CSV files.

Future work

- Adding more coumns in existing dataset like genre, number of votes and cast.
- Improving the documentation part of the project.
- Creating a dataset containing different genres of series, each CSV file will conatin a certain genre.

Reference links

- https://www.imdb.com/
- https://www.imdb.com/search/title/?at=0&genres=animation&keywords=anime&num_votes=1000,&sort=user_rating&title_type=tv_series
- https://www.imdb.com/search/title/?title_type=tv_series&num_votes=1000,&genres=animation&keywords=anime&sort=user_rating,desc&start=5

```
jovian.commit(project="scraping-top-imdb-animated-series")
```

```
```