- INFO 215: Web Science. Lecture 03: Web Scraping
- 30 January, 2022

- *Fazle Rabbi, Ph.D*
- *Assoc. Prof in Information Science*
- *University of Bergen*
- *Fazle.Rabbi@uib.no*
- *Bergen, Norway*

Learning Objectives:

Theoretical knowledge:
What is Web Scraping? What are the benefits of Web Scraping ?
Basics of parsing.
Search patterns and regular expressions.

Practical knowledge:
How to parse html files using Python libraries?
Getting familiar with BeautifulSoup
How to handle exceptions while parsing html files?
How to navigate DOM tree using BeautifulSoup

What is Web Scraping?

Web “scraping” (also called “web harvesting,” “web data extraction,” or even “web data mining”), can be defined as “the construction of an agent to download, parse, and organize data from the web in an automated manner.”

Simply speaking, Web scraping is the process of gathering information from the Internet.

What are the benefits of Web Scraping?

Availability of data:

- The web contains lots of interesting data sources that provide a treasure trove for all sorts of interesting things.
- The current unstructured nature of the web does not always make it easy to gather or export this data in an easy manner.
- Instead of viewing the information in web page by page, web scraping allows us to automatically gather a rich data set from the web.

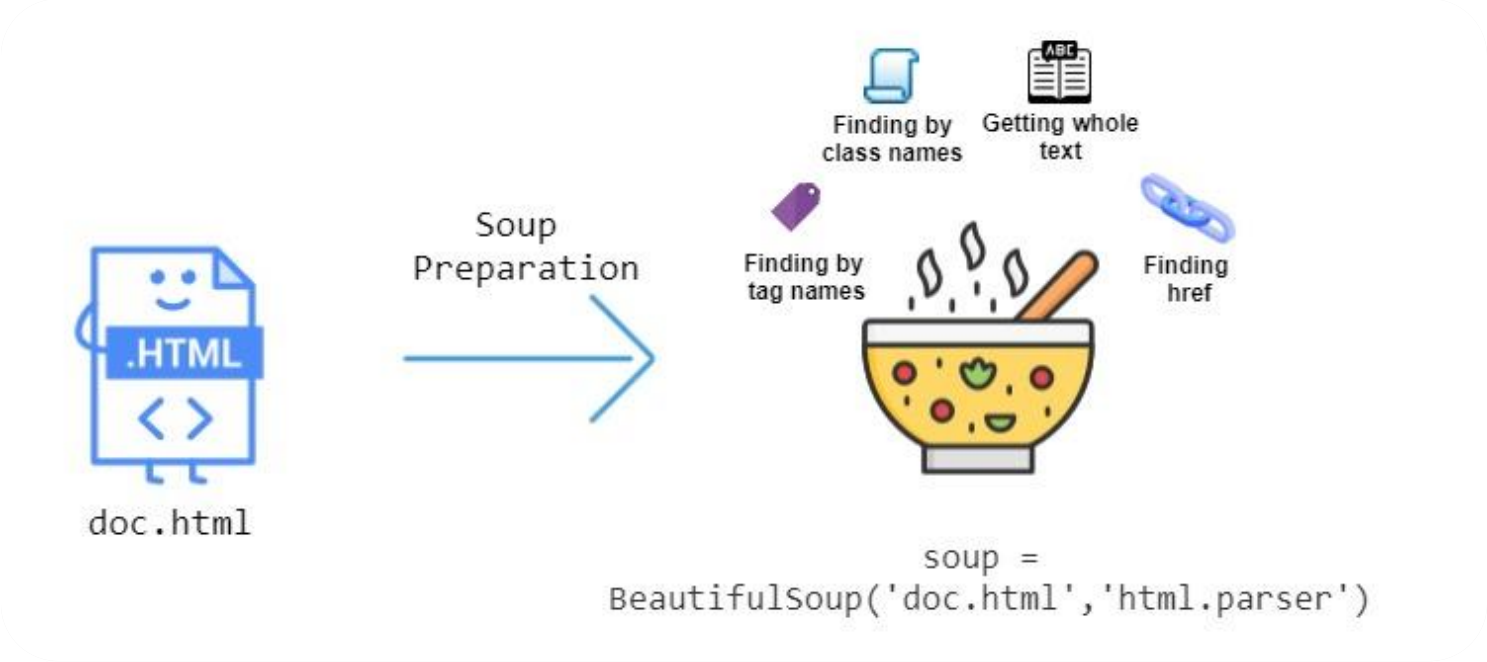
What are the benefits of Web Scraping?

Reasons why web scraping might be preferable over the use of an API:

- The website you want to extract data from does not provide an API.
- The API provided is not free (whereas the website is).
- The API provided is rate limited: meaning you can only access it a number of certain times per second, per day, ...
- The API does not expose all the data you wish to obtain (whereas the website does).

Getting Familiar with BeautifulSoup

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree.



https://stackabuse.com/guide-to-parsing-html-with-beautifulsoup-in-python/

Getting Familiar with BeautifulSoup

The complete instructions for installing BeautifulSoup 4 can be found at Crummy.com

Installing in Linux:
$ sudo apt-get install python-bs4

Installing in Mac:
pip3 install beautifulsoup4

Installing in Windows:
Download the most recent BeautifulSoup 4 release from (https://www.crummy.com/software/BeautifulSoup/#Download) page, navigate to the directory you unzipped it to, and run this:
> python setup.py install

Types of parsers

Parser	Typical usage	Advantages	Disadvantages
Python's html.parser	BeautifulSoup(markup, "html.parser")	• Moderately fast • Lenient (As of Python 2.7.3 and 3.2.)	• Not as fast as lxml, less lenient than html5lib.
lxml's HTML parser	BeautifulSoup(markup, "lxml")	• Very fast • Lenient	• External C dependency
lxml's XML parser	BeautifulSoup(markup, "lxml-xml") BeautifulSoup(markup, "xml")	• Very fast • The only currently supported XML parser	• External C dependency
html5lib	BeautifulSoup(markup, "html5lib")	• Extremely lenient • Parses pages the same way a web browser does • Creates valid HTML5	• Very slow • External Python dependency

html.parser is included with Python 3 and requires no extra installations in order to use.
lxml parser can be installed through pip: $ pip3 install lxml

https://en.wikipedia.org/wiki/Beautiful_Soup_(HTML_parser)
https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Getting Familiar with BeautifulSoup

Three features make it powerful:

- Beautiful Soup provides a few simple methods and idioms for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need. It doesn't take much code to write an application

- Beautiful Soup automatically converts incoming documents to Unicode and outgoing documents to UTF-8. You don't have to think about encodings, unless the document doesn't specify an encoding and Beautiful Soup can't detect one. Then you just have to specify the original encoding.

- Beautiful Soup sits on top of popular Python parsers like lxml and html5lib, allowing you to try out different parsing strategies or trade speed for flexibility.

https://www.crummy.com/software/BeautifulSoup/#Download

Getting familiar with BeautifulSoup.

```
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('http://www.pythonscraping.com/pages/page1.html')
bs = BeautifulSoup(html.read(), 'html.parser')
print(bs.h1)
```

The output is as follows:

```
<h1>An Interesting Title</h1>
```

Troubleshooting tips for MacOs users.

If you get an error related to SSL certificate in MacOS, try the following commands in terminal window:

```
pip3 install certifi
/Applications/Python\ 3.10/Install\ Certificates.command
```

Note that you may have a different version than 3.10 in your setup so you have to use the related version number in the above command.

Getting familiar with BeautifulSoup.

http://www.pythonscraping.com/pages/page1.html

```html
<html>
<head>
<title>A Useful Page</title>
</head>
<body>
<h1>An Interesting Title</h1>
<div>
Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.
</div>
</body>
</html>
```

By executing this code:
bs = BeautifulSoup(html, 'html.parser')

This HTML content is transformed into a BeautifulSoup object, with the following structure:

- **html** → *<html><head>...</head><body>...</body></html>*
  - — **head** → *<head><title>A Useful Page<title></head>*
    - — **title** → *<title>A Useful Page</title>*
  - — **body** → *<body><h1>An Int...</h1> Lorem ip... </body>*
    - — **h1** → *<h1>An Interesting Title</h1>*
    - — **div** → * Lorem Ipsum dolor... *

Getting familiar with BeautifulSoup.

```python
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('http://www.pythonscraping.com/pages/page1.html')
bs = BeautifulSoup(html.read(), 'html.parser')
print(bs.h1)
```

The output is as follows:

```
<h1>An Interesting Title</h1>
```

- **html** → *<html><head>...</head><body>...</body></html>*
  - — **head** → *<head><title>A Useful Page<title></head>*
    - — **title** → *<title>A Useful Page</title>*
  - — **body** → *<body><h1>An Int...</h1> Lorem ip... </body>*
    - — **h1** → *<h1>An Interesting Title</h1>*
    - — **div** → * Lorem Ipsum dolor... *

Why print(bs.h1) produces this output?

From the structure we see that h1 tag is nested two layers deep into the beautifulsoup object structure!

Getting familiar with BeautifulSoup.

```python
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('http://www.pythonscraping.com/pages/page1.html')
bs = BeautifulSoup(html.read(), 'html.parser')
print(bs.h1)
```

The output is as follows:

```
<h1>An Interesting Title</h1>
```

- **html** → *<html><head>...</head><body>...</body></html>*
  - — **head** → *<head><title>A Useful Page<title></head>*
    - — **title** → *<title>A Useful Page</title>*
  - — **body** → *<body><h1>An Int...</h1> Lorem ip... </body>*
    - — **h1** → *<h1>An Interesting Title</h1>*
    - — **div** → * Lorem Ipsum dolor... *

Why print(bs.h1) produces this output?

From the structure we see that h1 tag is nested two layers deep into the beautifulsoup object structure!

In fact, any of the following function calls would produce the same output:

bs.html.body.h1
bs.body.h1
bs.html.h1

Exception handling:

Many things can go wrong while accessing data from web:
- The website you are accessing, it can go down.
- The webpage you are parsing, it could be poorly formatted.
- The webpage you are parsing, it could have missing closing tags.

Example,
Two things can go wrong in the following line:

```
html = urlopen('http://www.pythonscraping.com/pages/page1.html')
```

- The page is not found on the server (or there was an error in retrieving it).
- The server is not found.

How to handle this situation ?

Exception handling:

Example,
html = urlopen('http://www.pythonscraping.com/pages/page1.html')

• The page is not found on the server (or there was an error in retrieving it).
> In this case, the urlopen function will throw a generic exception HTTPError.
This HTTP error may be “404 Page Not Found,” “500 Internal Server Error,” and so forth.

• The server is not found.
> If the server is not found at all (if, say, *http://www.pythonscraping.com* is down, or the URL is mistyped), urlopen will throw an URLError.

Exception handling:

```python
from urllib.request import urlopen
from urllib.error import HTTPError
from bs4 import BeautifulSoup


try:
    html = urlopen('http://www.pythonscrapingg.com/pages/page5.html')
except HTTPError as e:
    print(e)
else:
    bs = BeautifulSoup(html.read(), 'html.parser')
    print(bs.h1)
```

Analyze this python code.
Do you see any issue in the code ?
How do you fix the problem?

Exception handling:

```python
from urllib.request import urlopen
from urllib.error import HTTPError
from urllib.error import URLError
from bs4 import BeautifulSoup

try:
    html = urlopen('http://www.pythonscraping.com/pages/page1.html')
except HTTPError as e:
    print(e)
except URLError as e:
    print(e)
else:
    bs = BeautifulSoup(html.read(), 'html.parser')
    print(bs.div.a.getText())
```

Analyze this python code.
Do you see any issue in the code ?
How do you fix the problem?

Exception handling:

```python
from urllib.request import urlopen
from urllib.error import HTTPError
from bs4 import BeautifulSoup

def getTitle(url):
    try:
        html = urlopen(url)
    except HTTPError as e:
        return None
    try:
        bs = BeautifulSoup(html.read(), 'html.parser')
        title = bs.body.h1
    except AttributeError as e:
        return None
    return title

title = getTitle('http://www.pythonscraping.com/pages/page1.html')
if title == None:
    print('Title could not be found')
else:
    print(title)
```

Analyze this python code.
Do you see any issue in the code ?
How do you fix the problem?

Every time you access a tag in a BeautifulSoup object, it's smart to add a check to make sure the tag actually exists.
If you attempt to access a tag that does not exist, BeautifulSoup will return a None object.
The problem is, attempting to access a tag or a method on a None object will result in an AttributeError being thrown.

Searching with BeautifulSoup

find() / find_all()
find_parent() / find_parents()
find_next_sibling() / find_next_siblings()
find_previous_sibling() / find_previous_siblings()
find_next() / find_all_next()
find_previous() / find_all_previous()

Searching with BeautifulSoup

- find(tag, attributes, recursive, text, keywords);
- find_all(tag, attributes, recursive, text, limit, keywords).

About the parameters:

- tag: you can pass a string name of a tag or even a Python list of string tag names.
- attributes: you can pass a Python dictionary of attributes that matches tags that contain any one of those attributes
- recursive: you can pass a boolean to tell how deeply into the document you want to search. If recursive is set to True, the find_all function looks into children, and children's children
- text: you can pass a string argument to match based on the text content of the tags
- limit: you can set this with an integer x if you're interested only in retrieving the first *x* items from the page
- keywords: The keyword argument allows you to select tags that contain a particular attribute or set of attributes.

Note: find() and find_all() functions also accept a regular expression instead of a string.

Searching with BeautifulSoup

```python
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('http://www.pythonscraping.com/pages/page3.html')
bs = BeautifulSoup(html, 'html.parser')

for child in bs.find('table',{'id':'giftList'}).children:
    print(child)
```

What will this code print out ?

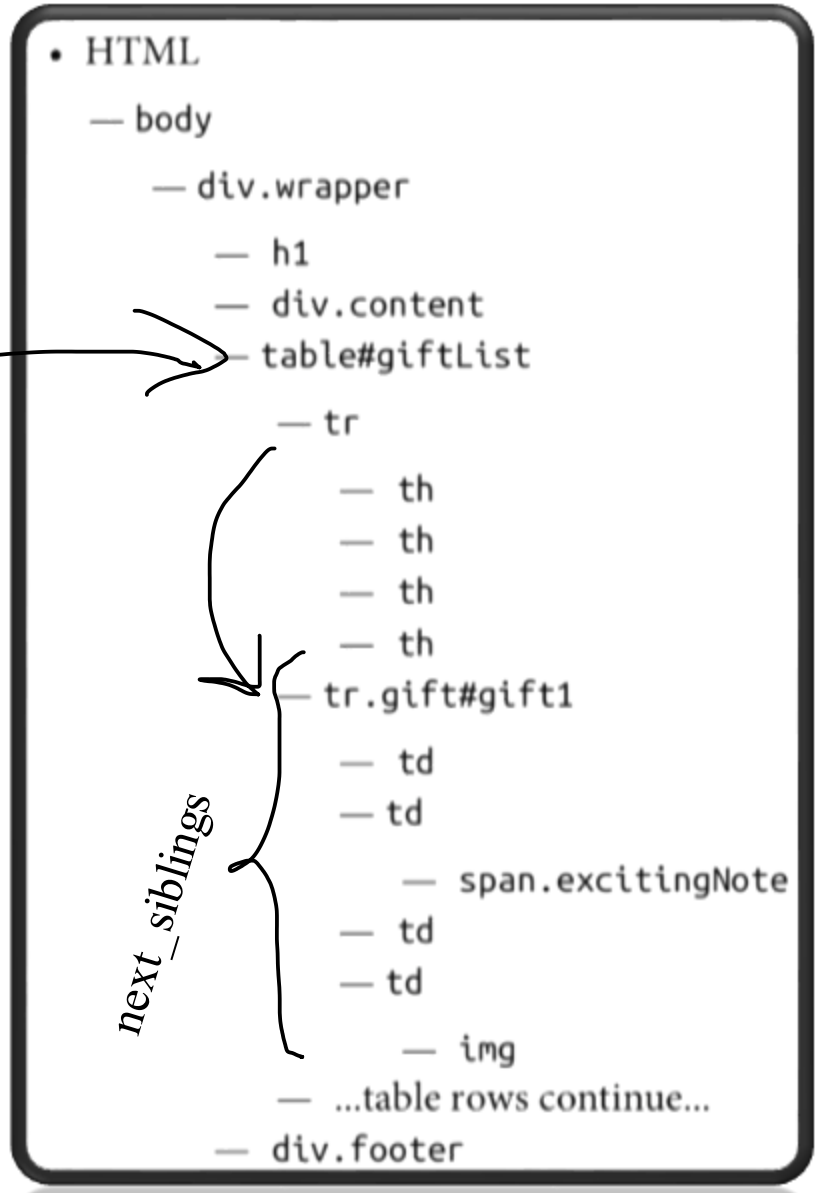
- HTML
  - — body
    - — div.wrapper
      - — h1
      - — div.content
      - — table#giftList
        - — tr
          - — th
          - — th
          - — th
          - — th
        - — tr.gift#gift1
          - — td
          - — td
            - — span.excitingNote
          - — td
          - — td
            - — img
        - — ...table rows continue...
      - — div.footer

Searching with BeautifulSoup

```python
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('http://www.pythonscraping.com/pages/page3.html')
bs = BeautifulSoup(html, 'html.parser')

for sibling in bs.find('table', {'id':'giftList'}).tr.next_siblings:
    print(sibling)
```



What will this code print out ?

Searching with BeautifulSoup

```python
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('http://www.pythonscraping.com/pages/page3.html')
bs = BeautifulSoup(html, 'html.parser')
print(bs.find('img',
              {'src':'../img/gifts/img1.jpg'})
      .parent.previous_sibling.get_text())
```

What will this code print out ?

- <tr>
  - td
  - td
  - td ❸
    - "$15.00" ❹
  - td ❷
    - <img src="../img/gifts/img1.jpg"> ❶

❶ The image tag where `src="../img/gifts/img1.jpg"` is first selected.

❷ You select the parent of that tag (in this case, the `td` tag).

❸ You select the `previous_sibling` of the `td` tag (in this case, the `td` tag that contains the dollar value of the product).

❹ You select the text within that tag, "$15.00."

Search patterns and regular expressions

A regular expression (regex) defines a sequence of patterns (an expression) defining a search pattern.
It is frequently used for string searching and matching code to find (and replace) fragments of strings.
There are often times when you want to quickly match tags and tag content in a web page. Regular expressions are a good tool to do this.

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
*	Matches the preceding character, subexpression, or bracketed character, 0 or more times.	a*b*	aaaaaaaa, aaabbbbb, bbbbbb, pppppp

Search patterns and regular expressions

There are often times when you want to quickly match tags and tag content in a web page. Regular expressions are a good tool to do this.

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
+	Matches the preceding character, subexpression, or bracketed character, 1 or more times.	a+b+	aaaaaaaab, aaabbbbb, abbbbbb

Click to add text

Is this a match for a+b+ regular expression?
abbbbb ---> this is a match here.
Abbbbb ----> this is not a match

Search patterns and regular expressions

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
[]	Matches any character within the brackets (i.e., "Pick any one of these things").	[A-Z]*	APPLE, CAPITALS, QWERTY

Search patterns and regular expressions

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
()	A grouped subexpression (these are evaluated first, in the "order of operations" of regular expressions).	(a*b)*	aaabaab, abaaab, ababaaaaab

Search patterns and regular expressions

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
{m, n}	Matches the preceding character, subexpression, or bracketed character between *m* and *n* times (inclusive).	a{2,3}b{2,3}	aabbb, aaabbb, aabb

Search patterns and regular expressions

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
[^]	Matches any single character that is *not* in the brackets.	[^A-Z]*	apple, lowercase, qwerty

Search patterns and regular expressions

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
|	Matches any character, string of characters, or subexpression, separated by the `I` (note that this is a vertical bar, or *pipe*, not a capital i).	b(a|i|e)d	bad, bid, bed

Search patterns and regular expressions

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
.	Matches any single character (including symbols, numbers, a space, etc.).	b.d	bad, bzd, b$d, b d

Search patterns and regular expressions

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
^	Indicates that a character or subexpression occurs at the beginning of a string.	^a	apple, asdf, a

Patpattern is ^h

Example match: h1, h2, h3,

Search patterns and regular expressions

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
\	An escape character (this allows you to use special characters as their literal meanings).	\^ \\| \\	^ | \

Search patterns and regular expressions

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
$	Often used at the end of a regular expression, it means "match this up to the end of the string." Without it, every regular expression has a de facto ".*" at the end of it, accepting strings where only the first part of the string matches. This can be thought of as analogous to the ^ symbol.	[A-Z]*[a-z]*$	ABCabc, zzzyx, Bob

Search patterns and regular expressions

Table 2-1. Commonly used regular expression symbols

Symbol(s)	Meaning	Example	Example matches
?!	"Does not contain." This odd pairing of symbols, immediately preceding a character (or regular expression), indicates that that character should not be found in that specific place in the larger string. This can be tricky to use; after all, the character might be found in a different part of the string. If trying to eliminate a character entirely, use in conjunction with a ^ and $ at either end.	^((?![A-Z]).)*$	no-caps-here, $ymb0ls a4e f!ne

More on regular expressions

Characters/Symbol	**Meaning**
abc..	Letters
123…	Digits
\d	Any Digit
\D	Any Non-digit character
.	Any Character
\.	Period
[abc]	Only a, b, or c
[^abc]	Not a, b, nor c
[a-z]	Characters a to z
[0-9]	Numbers 0 to 9

More on regular expressions

Characters/Symbol	**Meaning**
\w	Any Alphanumeric character
\W	Any Non-alphanumeric character
{m}	m Repetitions
{m,n}	m to n Repetitions
*	Zero or more repetitions
+	One or more repetitions
?	Optional character
\s	Any Whitespace
\S	Any Non-whitespace character
^…$	Starts and ends
(…)	Capture Group
(a(bc))	Capture Sub-group
(.*)	Capture all
(abc|def)	Matches abc or def

Example: Regular expression

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

html = urlopen('http://www.pythonscraping.com/pages/page3.html')
bs = BeautifulSoup(html, 'html.parser')
images = bs.find_all('img',
    {'src':re.compile('..\/img\/gifts/img.*.jpg')})
for image in images:
    print(image['src'])
```

What will this code print out ?

More on find() and find_all() function.

Both find() and find_all() return Tag objects.

- We can retrieve following information from a Tag object:
- Access the name attribute to retrieve the tag name.
- Access the contents attribute to get a Python list containing the tag's children (its direct descendant tags) as a list.
- The children attribute does the same but provides an iterator instead; the descendants attribute also returns an iterator,
- Similarly, you can also go "up" the HTML tree by using the parent and parents attributes. To go sideways (i.e., find next and previous elements at the same level in the hierarchy), next_sibling, previous_sibling and next_siblings, and previous_siblings can be used.
- Access the attributes of the element through the attrs attribute of the Tag object. For the sake of convenience, you can also directly use the Tag object itself as a dictionary.
- Use the text attribute to get the contents of the Tag object as clear text (without HTML tags).
- Alternatively, you can use the get_text method
- Finally, not all find and find_all searches need to start from your original BeautifulSoup objects. Every Tag object itself can be used as a new root from which new searches can be started.

What this code will print out?

```python
from urllib.request import urlopen
from bs4 import BeautifulSoup

url = 'https://en.wikipedia.org/w/index.php?title=List_of_Game_of_Thrones_episodes&oldid=802553687'
html = urlopen(url)
html_soup = BeautifulSoup(html.read(), 'html.parser')
# Find the first h1 tag
first_h1 = html_soup.find('h1')
print(first_h1.name)
print(first_h1.contents)
print(str(first_h1))

print(first_h1.text)
print(first_h1.attrs)

print(first_h1.attrs['id'])
```

Extract following information from a wiki page:

- In this assignment you will extract information from the following wiki page: https://en.wikipedia.org/wiki/Star_Wars:_The_Rise_of_Skywalker

- You have to extract all links from the page as well as where they point to (tip: look for the “href” attribute in “<a>” tags).

- You have to extract all images src attribute from the page.

- You have to extract all the awards that the movie has won. This one is a bit tricky. You have to print out all the information (I.e., Award, Date of ceremony, Category, Recipient(s), Result, Ref.) of the row that contains 'Won'. (tip: you may use the following code to select the td tag that has 'Won' value: bs.find_all('td', {'class':'yes table-yes2'}))

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

20 years of WIKIPEDIA Over One Billion Edits

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page

Star Wars: The Rise of Skywalker

From Wikipedia, the free encyclopedia

"Episode IX" redirects here. For other uses, see Episode 9.

Star Wars: The Rise of Skywalker (also known as ***Star Wars: Episode IX – The Rise of Skywalker***) is a 2019 American epic space opera film produced, co-written, and directed by J. J. Abrams. Produced by Lucasfilm and Abrams' production company Bad Robot Productions, and distributed by Walt Disney Studios Motion Pictures, it is the third installment of the *Star Wars* sequel trilogy, following *The Force Awakens* (2015) and *The Last Jedi* (2017), and the final episode of the nine-part "Skywalker saga".[a] The film's ensemble cast includes Carrie Fisher,[b] Mark Hamill, Adam Driver, Daisy Ridley, John Boyega, Oscar Isaac, Anthony Daniels, Naomi Ackie, Domhnall Gleeson, Richard E. Grant, Lupita Nyong'o, Keri Russell, Joonas Suotamo, Kelly Marie Tran, Ian McDiarmid, and Billy Dee Williams.

The Rise of Skywalker follows Rey, Finn, and Poe Dameron as they lead the Resistance's final stand against Supreme Leader Kylo Ren and the First Order, who are aided by the return of the deceased galactic emperor, Palpatine. Following initial reports that *The Last Jedi* director Rian Johnson would write the script for *Episode IX*, in August 2015, Colin Trevorrow was hired to direct and to write a script with his collaborator Derek Connolly; both ultimately retain story credit with Abrams and Chris Terrio. In September 2017, Trevorrow left the project following creative differences with producer Kathleen Kennedy, and Abrams returned as director. John Williams, composer for the previous episodic films, returned to compose the score–his final score for the franchise.[7] Principal photography began in August 2018 at Pinewood Studios in England and wrapped in February 2019. Post-production was completed on November 24, 2019. With an estimated budget of $275 million, it is one of the most expensive films ever made.

The film premiered in Los Angeles on December 16, 2019, and was released in the United States on December 20. It received mixed reviews from critics, who praised the acting, action sequences, musical score, and visual effects, but criticized the story, pacing, and its perceived departures from the plot and themes of *The Last Jedi*. The film grossed over $1.074 billion worldwide, becoming the seventh-highest-grossing film of 2019; although

Star Wars: The Rise of Skywalker

Theatrical release poster

Obligatory Assignment 02. (Due date: TBA)

Useful links:

- https://www.w3resource.com/python-exercises/web-scraping/index.php#EDITOR
- https://regexone.com/