Mandatory Assignment 5

Regular Expressions & Web Scraping (30 + 15 Points/40 + 5 Points)

University of Oslo - IN3110/IN4110

Fall 2020

Your solution to this mandatory assignment needs to be placed in the directory `assignment5` in your Github repository. The repository has to contain a `README.md` file with information on how to run your scripts, required dependencies and packages (versions the code ran with) as well as how to install them. Documentation on how to run your examples[1] is required. Furthermore, your code needs to be well commented and documented. **All** functions need to have docstrings explaining what the function does, how it should be used, an explanation of the parameters and return value(s) (including types). We **highly** recommend you use a well-established docstring style such as the Google style docstrings[2]. However, you are free to choose your own docstring style. We expect your code to be well formatted and readable. Coding style and documentation will be part of the point evaluation for **all tasks** in this assignment.

Files to Deliver for this Task

All `.py`-files will be delivered in the same `assignment5` folder. Each subtask will create an output folder with all outputs as stated in the subtask.

Files required for all tasks

- `README.md` including dependencies and their installation and commands you used for running your examples and creating your output files

Note: Regular expressions can be tricky to write, and they can get hairy very quickly. In addition to being difficult to write, regular expressions are even harder to read. Documentation for your regular expressions should also be given. You are also required to supply files that can be used to demonstrate your solutions.

- You are required to supply everything necessary to demonstrate your code, i.e. you should include the code examples you use when testing your work, or something equivalent.

[1] Not only important to get all of the credits, but it is a really important and not understated habit to be established :)

[2] `https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html`

5.1 Sending url requests (2 Points)

In order to parse html and become the regex master of the year, you need to access the html body first. This can be done using the python request module [3].

The module can be installed as follows:

```
pip install requests
```

To retrieve data from any destination, the most common approach is the **get** Request. This is the most used HTTP method. In our example, we will use the get request to fetch data from the website of choice. The result will be printed out as html data. The request module can be imported and used as shown below.

```
#import the module
import requests as req

#grabbing the content of https://en.wikipedia.org/wiki/URL
resp = req.get(”https://en.wikipedia.org/wiki/URL”)

#get() method returns a response object
print(resp.text)
<!DOCTYPE html>
<html class=”client−nojs” lang=”en” dir=”ltr”>
<head>
<meta charset=”UTF−8”/>
<title>URL − Wikipedia</title>

#shortened for the purpose of viewing
```

Fantastic, now we can actually display the data in html!

Often you are interested in a specific type of data. With GET requests, you can send parameters in the form of query strings. How does that work? If you were to construct the URL by hand, the data you are interested in would be given as key/value pairs in the URL after a question mark, e.g. httpbin.org/get?key=val. Using get requests, you can provide these arguments as a dictionary of strings, using the params keyword argument. That means, if you wanted to pass key1=value1 and key2=value2 to httpbin.org/get the code would in theory look like this:

```
import requests as req

params = {’key1’: ’value1’, ’key2’: ’value2’}
r = req.get(’https://httpbin.org/get’, params=params)

# the url would be encoded accordingly
print(r.url)
https://httpbin.org/get?key2=value2&key1=value1
```

[3] https://requests.readthedocs.io/en/master/user/install/

For a real example you can look at the following snippet:

```
import requests as req


params = {'user_name': 'admin', 'password': 'password'}
r = req.get('http://httpbin.org/get', params=params)

print(r.url)
http://httpbin.org/get?user_name=admin&password=password

# print the.text content of the request
print(r.text)

{
  "args": {
    "password": "password",
    "user_name": "admin"
  },
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.22.0",
    "X-Amzn-Trace-Id": "Root=1-5f841066-0
    c0ea5084573be8a230feaea"
  },
  "origin": "51.175.234.27",
  "url": "http://httpbin.org/get?user_name=admin&password=
    password"
}
```

Your task will be to create the function `get_html` which makes a url request of a given website. You should test your function on the following websites:

- `https://en.wikipedia.org/wiki/Studio_Ghibli`
- `https://en.wikipedia.org/wiki/Star_Wars`
- `https://en.wikipedia.org/wiki/Dungeons_%26_Dragons`

Furthermore you should extend the function to optionally take in parameters to be passed to the get function. There should be an optional argument allowing to specify that the response url and text will be saved to a text file with a specified name `optionalargument.txt`. If the optional argument is not set, the response is just returned.

The function should be able to be used like this:

```
get_html(url, params, output)
```

```
# if specified , the response txt and url get printed to a .
    txt - file with the name output.txt
```

The extended function with parameters should be tested for these websites:

- `https://en.wikipedia.org/w/index.php`
  - with parameters title=Main_Page and action=info
- `https://en.wikipedia.org/w/index.php`
  - with parameters title=Hurricane_Gonzalo and oldid=983056166

Files to Deliver in this Subtask

Create a folder `requesting_urls` where you put all output files from our test runs created for solving this subtask.

Files Required in this Subtask

- `requesting_urls.py`
- Five files inside the `requesting_urls` folder containing the function output for the five websites example websites.

5.2 Regex for filtering URLs (5 Points)

In this task, you will be making functions for finding urls in a body of html using regex. Create a script named `filter_urls.py` that includes a function `find_urls` that receives a string of html and returns a list of all urls found in the text. You will be using the `re.findall` method taught in the lecture. This will give us a list of all the strings matching your regular expression. To be strict, we only consider urls that are anchor[4] hyperlinks with the `<a>` HTML tag. The function should ignore fragment identifiers. That is, links that start with a hash (#) symbol. If the url points to a different page, but a specific fragment/section using the hash symbol, the fragment part of the url should be stripped before it is returned by the function.

Example: The string `https://www.example.com/somepage#someidentifier` becomes `https://www.example.com/somepage`

In order to handle relative urls, the function could optionally receive a base url.

Further, make a function `find_articles` that calls `find_urls` and returns only urls to Wikipedia articles. This function should also use regex and **be able to handle any chosen language** (`no.wikipedia.org`, `en.wikipedia.org` etc).

Note: Non wikipedia pages can be ignored. Combine these functions with your functions from previous tasks in order to test them on the following websites:

[4] name of the HTML-element, with the HTML-tag ”a”

- `https://en.wikipedia.org/wiki/Nobel_Prize`
- `https://en.wikipedia.org/wiki/Bundesliga`
- `https://en.wikipedia.org/wiki/2019%E2%80%9320_FIS_Alpine_Ski_World_Cup`

Note 1: You should only return normal Wikipedia articles, so no special namespace article (or files). While you could exclude just the specific reserved namespaces [5], it is sufficient to exclude all articles with a colon. This will exclude some actual articles (for example `https://en.wikipedia.org/wiki/Avengers:_Endgame`, but we won't deduce points for it.

Note 2: You should only find urls that are in the href attribute of the a tag. It is *not always* the first attribute which means matching just ¡a href= is not enough, but you can assume some things about the use of special html characters since regex cannot parse html alone.

Note 3: You need to include relative urls, best be combined with the base url. Examples of relative urls: `/wiki/Executive_director` and `//en.wikipedia.org/wiki/Wikipedia:Contact_us`

Note 4¿ The outputs for both first `fins_urls` and `fins_articles` should be added to your outputs.

The found urls should be saved to a .txt file, which name can be defined by an optional argument. **Files to Deliver in this Subtask**

Create a folder `filter_urls` where you put all output files from our test runs created for solving this subtask.

Files Required in this Subtask

- `filter_urls.py`
- Three files inside the `filter_urls` folder containing a list of the urls returned for each of the three example websites.

Note: You should not use Beautiful Soup or any HTML parser in this task, only regex.

5.3 Regular Expressions for finding Dates (IN3110 OPTIONAL & IN4110 REQUIRED 10 Points)

In this task, you will be making functions for finding "dates" in a body of html using regex. Create a script named `collect_dates.py` that includes a function `find_dates` that receives a string of html and returns a list of all dates found in the text in the following format:

```
1998/10/12
1998/11/04
1999/01/13
```

[5] i.e. look here `https://en.wikipedia.org/wiki/Wikipedia:Namespace`

Which date formats do we need to consider? You need to consider the wikipedia standardized formats [6]. These are the four formats:

```
DMY: 13 October 2020
MDY: October 13, 2020
YMD: 2020 October 13
ISO: 2020−10−13
```

Note: You also need to include the case, where the month is abbreviated, e.g. Oct for October, which adds these date types to consider.

```
DMY: 13 Oct 2020
MDY: Oct 13, 2020
YMD: 2020 Oct 13
```

We are aware, that Wikipedia should not support the abbreviated month, but we would like to allow your code to be flexible, in case you are handling other websites in the future

The function should furthermore be able to **recognize, if there is no day**, but only the month and the year. Then it should just return the year and the months. This optional day argument should work for the following formats resulting from the four formats you are working on:

```
DMY: October 2020
MDY: October , 2020
```

Since `2020-10` and `2020 October` can result in many false positives, there will **NOT** be points withdrawn if for that one if the missing date version is **NOT** implemented.

For a list of dates found which have two dates with `year/month/day` and one with `year/month` the returned list could look like this:

```
1) 1998/10/12
2) 1998/11
3) 1999/01/13
```

This is consequently the complete list of formats to consider:

```
DMY: 13 October 2020 AND October 2020
MDY: October 13, 2020 AND October , 2020
YMD: 2020 October 13
ISO: 2020−10−13
```

The dates **returned** should be formatted like this `year/month/day`.

Moreover, there should be an optional argument allowing to specify that the resulting list will be saved to a text file with a specified name `optionalargument.txt`.

You will be using the `re.findall` and `re.sub` method taught in the lecture. This will give us a list of all the strings matching your regular expressio and allow to change the formatting.

Here is the list of websites, you need to deliver a report in form of a text-file for:

Steps for creating the script

[6] `https://en.wikipedia.org/wiki/Template:Date`

- visit the following websites
- `https://en.wikipedia.org/wiki/Linus_Pauling`
- `https://en.wikipedia.org/wiki/Rafael_Nadal`
- `https://en.wikipedia.org/wiki/J._K._Rowling`
- `https://en.wikipedia.org/wiki/Richard_Feynman`
- `https://en.wikipedia.org/wiki/Hans_Rosling`

Save your implementation to `collect_dates.py`.

Files to Deliver in this Subtask

Create a folder `filter_dates_regex` where you put all output files from our test runs created for solving this subtask.

Files Required in this Subtask

- `collect_dates.py`
- Five files inside the `filter_dates_regex` folder containing a list of found dates in sorted order, for each website given.

Note: You should not use Beautiful Soup or any HTML parser in this task, only regex.

5.4 Making your Life easier with Soup for filtering datetime objects (IN3110 & IN4110 8 Points)

Need to plan your watch parties for the upcoming skiing season? Why not send out all the dates and times to your friends that are as well into (watching) skiing? In this task you will use the Python tool Beautiful soup to parse data. You will extract the datetime objects representing the event ”date” , a regular expression fetching the ”venue” and the discipline ”type”.

You can install Beautiful Soup via:

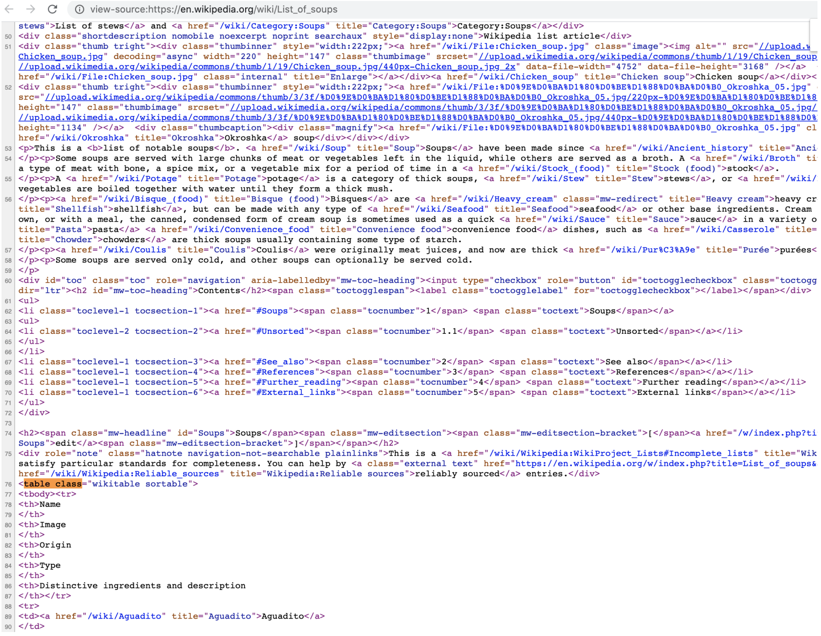
```
pip install beautifulsoup4
```

Note: Make sure that you are running python3 and your pip points to the right python version as well. If you are running into issues we highly recommend setting up a `virtualenv` or a `conda env`.

Before starting with the fun-coding part, it is recommended to take some time to get familiar with the html tags of the website we are extracting information from.

Note: It will really help for this task if you just take a few minutes to familiarize yourself with the page setup.

We recommend to navigate to the website using your favorite browser, right click and 'Inspect'. Get familiar with the elements used. Alternatively, right click on the web page and view the page source. Since we are interested in a table, search for the table class :



```
stews">List of stews</a> and <a href="/wiki/Category:Soups" title="Category:Soups">Category:Soups</a></div>
<div class="shortdescription nomobile noexcerpt noprint searchaux" style="display:none">Wikipedia list article</div>
<div class="thumb tright"><div class="thumbinner" style="width:222px;"><a href="/wiki/File:Chicken_soup.jpg" class="image"><img alt="" src="//upload.w
Chicken_soup.jpg" decoding="async" width="220" height="147" class="thumbimage" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/1/19/Chicken_soup
//upload.wikimedia.org/wikipedia/commons/thumb/1/19/Chicken_soup.jpg/440px-Chicken_soup.jpg 2x" data-file-width="4752" data-file-height="3168" /></a>
href="/wiki/File:Chicken_soup.jpg" class="internal" title="Enlarge"></a></div><a href="/wiki/Chicken_soup" title="Chicken soup">Chicken soup</a></div><
<div class="thumb tright"><div class="thumbinner" style="width:222px;"><a href="/wiki/File:%D0%9E%D0%BA%D1%80%D0%BE%D1%88%D0%BA%D0%B0_Okroshka_05.jpg"
src="//upload.wikimedia.org/wikipedia/commons/thumb/3/3f/%D0%9E%D0%BA%D1%80%D0%BE%D1%88%D0%BA%D0%B0_Okroshka_05.jpg/220px-%D0%9E%D0%BA%D1%80%D0%BE%D1%8
height="147" class="thumbimage" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/3/3f/%D0%9E%D0%BA%D1%80%D0%BE%D1%88%D0%BA%D0%B0_Okroshka_05.jpg/
//upload.wikimedia.org/wikipedia/commons/thumb/3/3f/%D0%9E%D0%BA%D1%80%D0%BE%D1%88%D0%BA%D0%B0_Okroshka_05.jpg/440px-%D0%9E%D0%BA%D1%80%D0%BE%D1%88%D0%
height="1134" /></a>  <div class="thumbcaption"><div class="magnify"><a href="/wiki/File:%D0%9E%D0%BA%D1%80%D0%BE%D1%88%D0%BA%D0%B0_Okroshka_05.jpg" cl
href="/wiki/Okroshka" title="Okroshka">Okroshka</a> soup</div></div></div>
<p>This is a <b>list of notable soups</b>. <a href="/wiki/Soup" title="Soup">Soups</a> have been made since <a href="/wiki/Ancient_history" title="Anci
</p><p>Some soups are served with large chunks of meat or vegetables left in the liquid, while others are served as a broth. A <a href="/wiki/Broth" ti
a type of meat with bone, a spice mix, or a vegetable mix for a period of time in a <a href="/wiki/Stock_(food)" title="Stock (food)">stock</a>.
</p><p>A <a href="/wiki/Potage" title="Potage">potage</a> is a category of thick soups, <a href="/wiki/Stew" title="Stew">stews</a>, or <a href="/wiki/
vegetables are boiled together with water until they form a thick mush.
</p><p><a href="/wiki/Bisque_(food)" title="Bisque (food)">Bisques</a> are <a href="/wiki/Heavy_cream" class="mw-redirect" title="Heavy cream">heavy cr
title="Shellfish">shellfish</a>, but can be made with any type of <a href="/wiki/Seafood" title="Seafood">seafood</a> or other base ingredients. Cream
own, or with a meal, the canned, condensed form of cream soup is sometimes used as a quick <a href="/wiki/Sauce" title="Sauce">sauce</a> in a variety o
title="Pasta">pasta</a> <a href="/wiki/Convenience_food" title="Convenience food">convenience food</a> dishes, such as <a href="/wiki/Casserole" title=
title="Chowder">chowders</a> are thick soups usually containing some type of starch.
</p><p><a href="/wiki/Coulis" title="Coulis">Coulis</a> were originally meat juices, and now are thick <a href="/wiki/Pur%C3%A9e" title="Purée">purées<
</p><p>Some soups are served only cold, and other soups can optionally be served cold.
</p>
<div id="toc" class="toc" role="navigation" aria-labelledby="mw-toc-heading"><input type="checkbox" role="button" id="toctogglecheckbox" class="toctogg
dir="ltr"><h2 id="mw-toc-heading">Contents</h2><span class="toctogglespan"><label class="toctogglelabel" for="toctogglecheckbox"></label></span></div>
<ul>
<li class="toclevel-1 tocsection-1"><a href="#Soups"><span class="tocnumber">1</span> <span class="toctext">Soups</span></a>
<ul>
<li class="toclevel-2 tocsection-2"><a href="#Unsorted"><span class="tocnumber">1.1</span> <span class="toctext">Unsorted</span></a></li>
</ul>
</li>
<li class="toclevel-1 tocsection-3"><a href="#See_also"><span class="tocnumber">2</span> <span class="toctext">See also</span></a></li>
<li class="toclevel-1 tocsection-4"><a href="#References"><span class="tocnumber">3</span> <span class="toctext">References</span></a></li>
<li class="toclevel-1 tocsection-5"><a href="#Further_reading"><span class="tocnumber">4</span> <span class="toctext">Further reading</span></a></li>
<li class="toclevel-1 tocsection-6"><a href="#External_links"><span class="tocnumber">5</span> <span class="toctext">External links</span></a></li>
</ul>
</div>

<h2><span class="mw-headline" id="Soups">Soups</span><span class="mw-editsection"><span class="mw-editsection-bracket">[</span><a href="/w/index.php?ti
Soups">edit</a><span class="mw-editsection-bracket">]</span></span></h2>
<div role="note" class="hatnote navigation-not-searchable plainlinks">This is a <a href="/wiki/Wikipedia:WikiProject_Lists#Incomplete_lists" title="Wik
satisfy particular standards for completeness. You can help by <a class="external text" href="https://en.wikipedia.org/w/index.php?title=List_of_soups&
href="/wiki/Wikipedia:Reliable_sources" title="Wikipedia:Reliable sources">reliably sourced</a> entries.</div>
<table class="wikitable sortable">
<tbody><tr>
<th>Name
</th>
<th>Image
</th>
<th>Origin
</th>
<th>Type
</th>
<th>Distinctive ingredients and description
</th></tr>
<tr>
<td><a href="/wiki/Aguadito" title="Aguadito">Aguadito</a>
</td>
```

Figure 1: Table class in the html. You get to this view using "View Source Page" mode.

Okay, so we would like to access the table of soups now. To do so we need to first grab the HTML from the web page, so that we have something to parse through. Then we will extract the soup table.

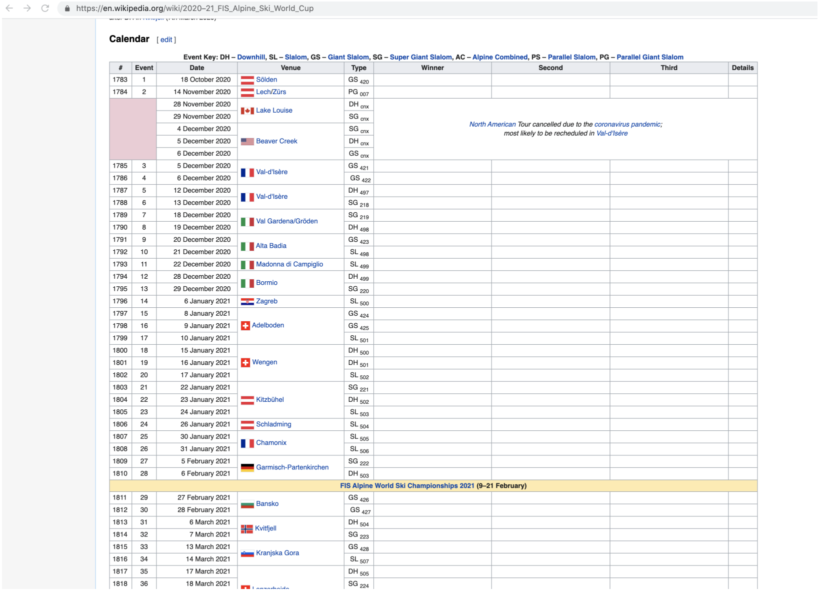
```
from bs4 import BeautifulSoup
import requests as req

url = "https://en.wikipedia.org/wiki/List_of_soups"

# request url as we have already learned - yeah!
request = req.get(url)

soup = BeautifulSoup(request.text, "html.parser")

#check the title of the wikipedia page
print(soup.title)

# get the soup table
soup_table = soup.find('table', {"class":'wikitable sortable
    '})
```

Great, now we need to know how to navigate through the table. To figure out which column holds which information, it is recommended to look at the header. Therefore, we will use the html `<th>` tag, which defines the header cell in an html table. On a sidenote, a html table has to kinds of cells: header cells created with `<th>` elements and data cells created with `<td>` elements.

To iterate through the data in the table, we first need to access the table rows. Table rows are defined by the `<tr>` tag. A `<tr>` element contains `<td>` or `<th>` elements. The table headers will be extracted via `<th>` elements,

Write a script `time_planner.py` that first requests the url from `https://en.wikipedia.org/wiki/2019%E2%80%9320_FIS_Alpine_Ski_World_Cup`. The request can be done with a function created in earlier tasks.

Then parses through the html using `Beautiful Soup` and finds the main table in the calendar section (see figure below).



https://en.wikipedia.org/wiki/2020–21_FIS_Alpine_Ski_World_Cup

Calendar [edit]

Event Key: DH – Downhill, SL – Slalom, GS – Giant Slalom, SG – Super Giant Slalom, AC – Alpine Combined, PS – Parallel Slalom, PG – Parallel Giant Slalom

#	Event	Date	Venue	Type	Winner	Second	Third	Details
1783	1	18 October 2020	Sölden	GS $_{420}$				
1784	2	14 November 2020	Lech/Zürs	PG $_{007}$				
		28 November 2020	Lake Louise	DH $_{cnx}$	*North American Tour cancelled due to the coronavirus pandemic;* *most likely to be recheduled in Val-d'Isère*			
		29 November 2020		SG $_{cnx}$				
		4 December 2020	Beaver Creek	SG $_{cnx}$				
		5 December 2020		DH $_{cnx}$				
		6 December 2020		GS $_{cnx}$				
1785	3	5 December 2020	Val-d'Isère	GS $_{421}$				
1786	4	6 December 2020		GS $_{422}$				
1787	5	12 December 2020	Val-d'Isère	DH $_{497}$				
1788	6	13 December 2020		SG $_{218}$				
1789	7	18 December 2020	Val Gardena/Gröden	SG $_{219}$				
1790	8	19 December 2020		DH $_{498}$				
1791	9	20 December 2020	Alta Badia	GS $_{423}$				
1792	10	21 December 2020		SL $_{498}$				
1793	11	22 December 2020	Madonna di Campiglio	SL $_{499}$				
1794	12	28 December 2020	Bormio	DH $_{499}$				
1795	13	29 December 2020		SG $_{220}$				
1796	14	6 January 2021	Zagreb	SL $_{500}$				
1797	15	8 January 2021	Adelboden	GS $_{424}$				
1798	16	9 January 2021		GS $_{425}$				
1799	17	10 January 2021		SL $_{501}$				
1800	18	15 January 2021	Wengen	DH $_{500}$				
1801	19	16 January 2021		DH $_{501}$				
1802	20	17 January 2021		SL $_{502}$				
1803	21	22 January 2021	Kitzbühel	SG $_{221}$				
1804	22	23 January 2021		DH $_{502}$				
1805	23	24 January 2021		SL $_{503}$				
1806	24	26 January 2021	Schladming	SL $_{504}$				
1807	25	30 January 2021	Chamonix	SL $_{505}$				
1808	26	31 January 2021		SL $_{506}$				
1809	27	5 February 2021	Garmisch-Partenkirchen	SG $_{222}$				
1810	28	6 February 2021		DH $_{503}$				
FIS Alpine World Ski Championships 2021 (9–21 February)								
1811	29	27 February 2021	Bansko	GS $_{426}$				
1812	30	28 February 2021		GS $_{427}$				
1813	31	6 March 2021	Kvitfjell	DH $_{504}$				
1814	32	7 March 2021		SG $_{223}$				
1815	33	13 March 2021	Kranjska Gora	GS $_{428}$				
1816	34	14 March 2021		SL $_{507}$				
1817	35	17 March 2021	Lenzerheide	DH $_{505}$				
1818	36	18 March 2021		SG $_{224}$				

Figure 2: Main table in calendar section.

Write a function `extract_events` that extracts the data in the date, venue and discipline column. The function will live in the script `time_planner.py`. Since you would like to gamble with your friends beforehand you will create a betting slip. That slip will contain and a table which columns represent the date, venue and discipline as well as a column for the tip of which athlete or country wins that competition. The empty betting slip should be created automatically by your script and saved to a new file named `betting_slip_empty.md` using Markdown [7] formatting.

Check that your script also works when accessing `https://en.wikipedia.org/wiki/2020%E2%80%9321_FIS_Alpine_Ski_World_Cup`.

Files to Deliver in this Subtask

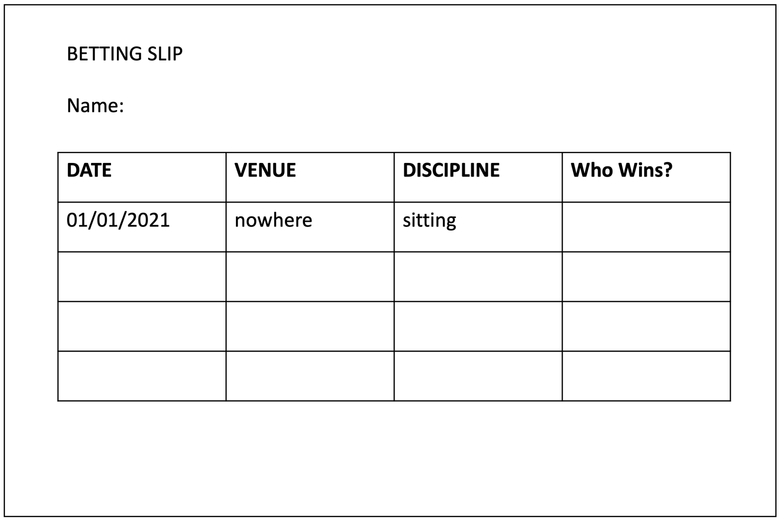
[7] `https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables`

Create a folder `datetime_filter` where you put all output files from our test runs created for solving this subtask. *Files Required in this Subtask*

- `time_planner.py`

- `datetime_filter/betting_slip_empty.md` - a nicely formatted table for your bets, containing date, venue and discipline, but with an empty "who wins" column.

The resulting betting slip would look like this:



BETTING SLIP

Name:

DATE	VENUE	DISCIPLINE	Who Wins?
01/01/2021	nowhere	sitting	

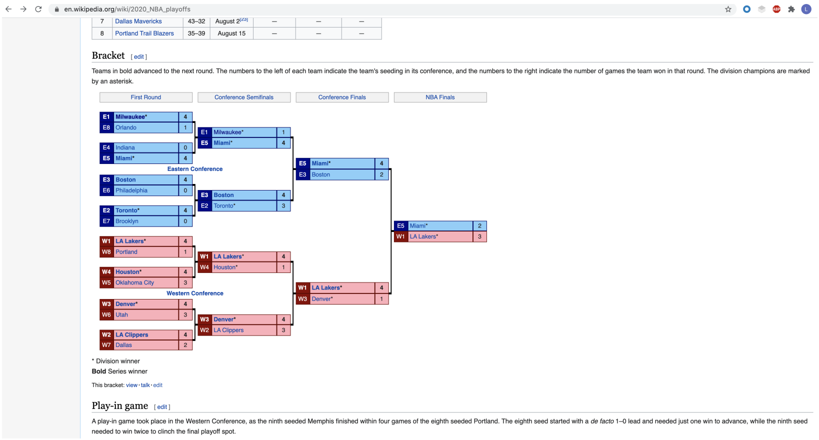
Figure 3: Betting Slip.

5.5 NBA Player Statistics Season 2019/2020 (IN3110 & IN4110 15 Points)

Go Miami Heat - Whooo! Let's imagine you are actually into basketball. After this unpredictable season, you really want dive into the player statistics. Lucky you! In this task you are going to write a script `fetch_player statistics.py` which visits the "2020 NBA playoffs" website on wikipedia (this one: `https://en.wikipedia.org/wiki/2020_NBA_playoffs`) and creates some player statistics. As in task `5.4` will use `BeautifulSoup` for parsing.

As always, you first need to make a url request using your function `get_html`.

Then you want to navigate, using `BeautifulSoup` to the main table in the "Bracket" section shown below.



7	Dallas Mavericks	43–32	August 2[23]	—	—	—
8	Portland Trail Blazers	35–39	August 15	—	—	—

Bracket [edit]

Teams in bold advanced to the next round. The numbers to the left of each team indicate the team's seeding in its conference, and the numbers to the right indicate the number of games the team won in that round. The division champions are marked by an asterisk.

First Round

E1 **Milwaukee*** 4
E8 Orlando 1

E4 Indiana 0
E5 **Miami*** 4

Eastern Conference

E3 **Boston** 4
E6 Philadelphia 0

E2 **Toronto*** 4
E7 Brooklyn 0

W1 **LA Lakers*** 4
W8 Portland 1

W4 **Houston*** 4
W5 Oklahoma City 3

Western Conference

W3 **Denver*** 4
W6 Utah 3

W2 **LA Clippers** 4
W7 Dallas 2

Conference Semifinals

E1 Milwaukee* 1
E5 **Miami*** 4

E3 **Boston** 4
E2 Toronto* 3

W1 **LA Lakers*** 4
W4 Houston* 1

W3 **Denver*** 4
W2 LA Clippers 3

Conference Finals

E5 **Miami*** 4
E3 Boston 2

W1 **LA Lakers*** 4
W3 Denver* 1

NBA Finals

E5 Miami* 2
W1 LA Lakers* 3

* Division winner
Bold Series winner

This bracket: view · talk · edit

Play-in game [edit]

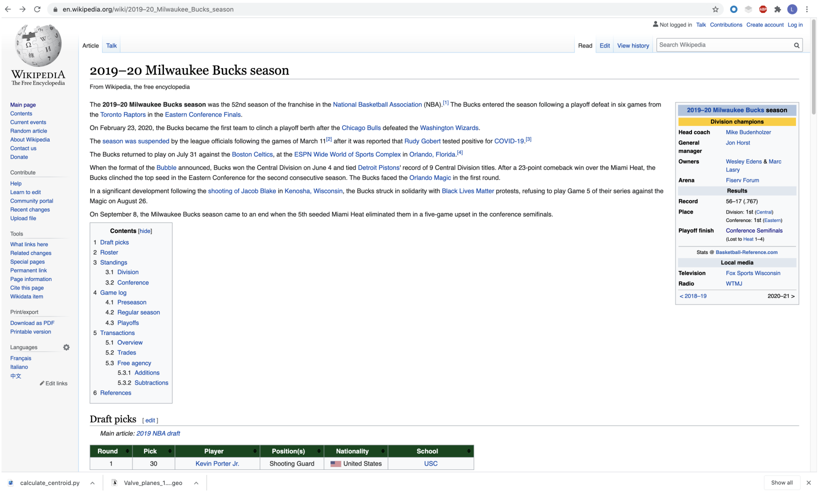
A play-in game took place in the Western Conference, as the ninth seeded Memphis finished within four games of the eighth seeded Portland. The eighth seed started with a *de facto* 1–0 lead and needed just one win to advance, while the ninth seed needed to win twice to clinch the final playoff spot.

Figure 4: Bracket Section.

There you will extract the names of the teams which made it to the conference semifinals (This is shown in the section ”Bracket”). [8]. You will also create a function `extract_url` which will allow you to extract the team urls in this table. You will follow the urls to the wikipedia websites of the corresponding teams in the Bracket section.

[8] Hint: It should be 8.

The next figure shows the website you get to when following the Milwaukee Bucks link in the conference semifinals.



2019–20 Milwaukee Bucks season

From Wikipedia, the free encyclopedia

The **2019–20 Milwaukee Bucks season** was the 52nd season of the franchise in the National Basketball Association (NBA).[1] The Bucks entered the season following a playoff defeat in six games from the Toronto Raptors in the Eastern Conference Finals.

On February 23, 2020, the Bucks became the first team to clinch a playoff berth after the Chicago Bulls defeated the Washington Wizards.

The season was suspended by the league officials following the games of March 11[2] after it was reported that Rudy Gobert tested positive for COVID-19.[3]

The Bucks returned to play on July 31 against the Boston Celtics, at the ESPN Wide World of Sports Complex in Orlando, Florida.[4]

When the format of the Bubble announced, Bucks won the Central Division on June 4 and tied Detroit Pistons' record of 9 Central Division titles. After a 23-point comeback win over the Miami Heat, the Bucks clinched the top seed in the Eastern Conference for the second consecutive season. The Bucks faced the Orlando Magic in the first round.

In a significant development following the shooting of Jacob Blake in Kenosha, Wisconsin, the Bucks struck in solidarity with Black Lives Matter protests, refusing to play Game 5 of their series against the Magic on August 26.

On September 8, the Milwaukee Bucks season came to an end when the 5th seeded Miami Heat eliminated them in a five-game upset in the conference semifinals.

Contents [hide]

1 Draft picks
2 Roster
3 Standings
  3.1 Division
  3.2 Conference
4 Game log
  4.1 Preseason
  4.2 Regular season
  4.3 Playoffs
5 Transactions
  5.1 Overview
  5.2 Trades
  5.3 Free agency
    5.3.1 Additions
    5.3.2 Subtractions
6 References

2019–20 Milwaukee Bucks season	
Division champions	
Head coach	Mike Budenholzer
General manager	Jon Horst
Owners	Wesley Edens & Marc Lasry
Arena	Fiserv Forum
Results	
Record	56–17 (.767)
Place	Division: 1st (Central) Conference: 1st (Eastern)
Playoff finish	Conference Semifinals (Lost to Heat 1–4)
Stats @ Basketball-Reference.com	
Local media	
Television	Fox Sports Wisconsin
Radio	WTMJ
< 2018–19	2020–21 >

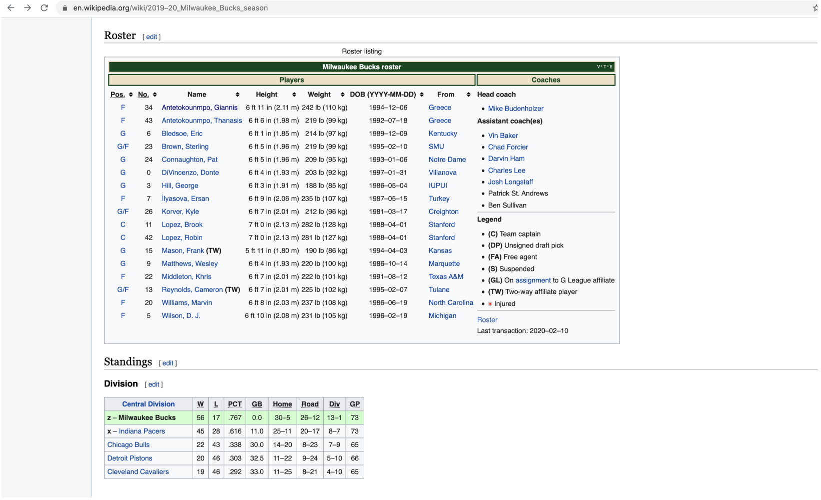
Draft picks [edit]

Main article: 2019 NBA draft

Round	Pick	Player	Position(s)	Nationality	School
1	30	Kevin Porter Jr.	Shooting Guard	United States	USC

Figure 5: Milwaukee Bucks Page.

In the next step you need to identify all the players that played for the specific team in that season. Therefore you need to navigate to the table holding the team roster. For the Milwaukee Bucks the roster section looks like this:



Roster [edit]

Roster listing

Milwaukee Bucks roster

Players							
Pos.	No.	Name	Height	Weight	DOB (YYYY-MM-DD)	From
F	34	Antetokounmpo, Giannis	6 ft 11 in (2.11 m)	242 lb (110 kg)	1994–12–06	Greece
F	43	Antetokounmpo, Thanasis	6 ft 6 in (1.98 m)	219 lb (99 kg)	1992–07–18	Greece
G	6	Bledsoe, Eric	6 ft 1 in (1.85 m)	214 lb (97 kg)	1989–12–09	Kentucky
G/F	23	Brown, Sterling	6 ft 5 in (1.96 m)	219 lb (99 kg)	1995–02–10	SMU
G	24	Connaughton, Pat	6 ft 5 in (1.96 m)	209 lb (95 kg)	1993–01–06	Notre Dame
G	0	DiVincenzo, Donte	6 ft 4 in (1.93 m)	203 lb (92 kg)	1997–01–31	Villanova
G	3	Hill, George	6 ft 3 in (1.91 m)	188 lb (85 kg)	1986–05–04	IUPUI
F	7	İlyasova, Ersan	6 ft 9 in (2.06 m)	235 lb (107 kg)	1987–05–15	Turkey
G/F	26	Korver, Kyle	6 ft 7 in (2.01 m)	212 lb (96 kg)	1981–03–17	Creighton
C	11	Lopez, Brook	7 ft 0 in (2.13 m)	282 lb (128 kg)	1988–04–01	Stanford
C	42	Lopez, Robin	7 ft 0 in (2.13 m)	281 lb (127 kg)	1988–04–01	Stanford
G	15	Mason, Frank **(TW)**	5 ft 11 in (1.80 m)	190 lb (86 kg)	1994–04–03	Kansas
G	9	Matthews, Wesley	6 ft 4 in (1.93 m)	220 lb (100 kg)	1986–10–14	Marquette
F	22	Middleton, Khris	6 ft 7 in (2.01 m)	222 lb (101 kg)	1991–08–12	Texas A&M
G/F	13	Reynolds, Cameron **(TW)**	6 ft 7 in (2.01 m)	225 lb (102 kg)	1995–02–07	Tulane
F	20	Williams, Marvin	6 ft 8 in (2.03 m)	237 lb (108 kg)	1986–06–19	North Carolina
F	5	Wilson, D. J.	6 ft 10 in (2.08 m)	231 lb (105 kg)	1996–02–19	Michigan

Coaches

Head coach

- Mike Budenholzer

Assistant coach(es)

- Vin Baker
- Chad Forcier
- Darvin Ham
- Charles Lee
- Josh Longstaff
- Patrick St. Andrews
- Ben Sullivan

Legend

- **(C)** Team captain
- **(DP)** Unsigned draft pick
- **(FA)** Free agent
- **(S)** Suspended
- **(GL)** On assignment to G League affiliate
- **(TW)** Two-way affiliate player
- ✚ Injured

Roster

Last transaction: 2020–02–10

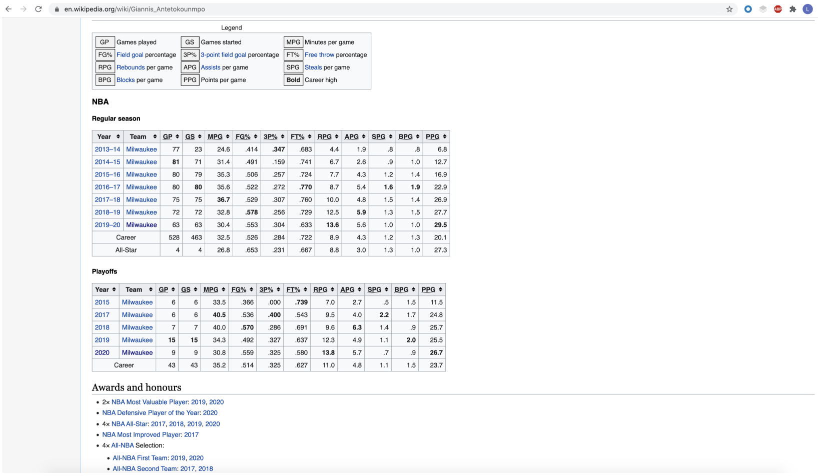
Standings [edit]

Division [edit]

Central Division	W	L	PCT	GB	Home	Road	Div	GP
z – Milwaukee Bucks	56	17	.767	0.0	30–5	26–12	13–1	73
x – Indiana Pacers	45	28	.616	11.0	25–11	20–17	8–7	73
Chicago Bulls	22	43	.338	30.0	14–20	8–23	7–9	65
Detroit Pistons	20	46	.303	32.5	11–22	9–24	5–10	66
Cleveland Cavaliers	19	46	.292	33.0	11–25	8–21	4–10	65

Figure 6: Roster Milwaukee Bucks.

Fantastic, let's start on the player statistic section! In order to filter the statistics for one specific player you need to extract the player name and the url to this players wikipedia page. You will request the `url` to that player which can be found in the roster. If we follow the link to Giannis Antetokounmpo this would be the part of the webpage showing the player statistics.



Legend

GP	Games played	GS	Games started	MPG	Minutes per game
FG%	Field goal percentage	3P%	3-point field goal percentage	FT%	Free throw percentage
RPG	Rebounds per game	APG	Assists per game	SPG	Steals per game
BPG	Blocks per game	PPG	Points per game	**Bold**	Career high

NBA

Regular season

Year	Team	GP	GS	MPG	FG%	3P%	FT%	RPG	APG	SPG	BPG	PPG
2013–14	Milwaukee	77	23	24.6	.414	**.347**	.683	4.4	1.9	.8	.8	6.8
2014–15	Milwaukee	**81**	71	31.4	.491	.159	.741	6.7	2.6	.9	1.0	12.7
2015–16	Milwaukee	80	79	35.3	.506	.257	.724	7.7	4.3	1.2	1.4	16.9
2016–17	Milwaukee	80	**80**	35.6	.522	.272	**.770**	8.7	5.4	**1.6**	**1.9**	22.9
2017–18	Milwaukee	75	75	**36.7**	.529	.307	.760	10.0	4.8	1.5	1.4	26.9
2018–19	Milwaukee	72	72	32.8	**.578**	.256	.729	12.5	**5.9**	1.3	1.5	27.7
2019–20	Milwaukee	63	63	30.4	.553	.304	.633	**13.6**	5.6	1.0	1.0	**29.5**
Career		528	463	32.5	.526	.284	.722	8.9	4.3	1.2	1.3	20.1
All-Star		4	4	26.8	.653	.231	.667	8.8	3.0	1.3	1.0	27.3

Playoffs

Year	Team	GP	GS	MPG	FG%	3P%	FT%	RPG	APG	SPG	BPG	PPG
2015	Milwaukee	6	6	33.5	.366	.000	**.739**	7.0	2.7	.5	1.5	11.5
2017	Milwaukee	6	6	**40.5**	.536	**.400**	.543	9.5	4.0	**2.2**	1.7	24.8
2018	Milwaukee	7	7	40.0	**.570**	.286	.691	9.6	**6.3**	1.4	.9	25.7
2019	Milwaukee	**15**	**15**	34.3	.492	.327	.637	12.3	4.9	1.1	**2.0**	25.5
2020	Milwaukee	9	9	30.8	.559	.325	.580	**13.8**	5.7	.7	.9	**26.7**
Career		43	43	35.2	.514	.325	.627	11.0	4.8	1.1	1.5	23.7

Awards and honours

- 2× NBA Most Valuable Player: 2019, 2020
- NBA Defensive Player of the Year: 2020
- 4× NBA All-Star: 2017, 2018, 2019, 2020
- NBA Most Improved Player: 2017
- 4× All-NBA Selection:
  - All-NBA First Team: 2019, 2020
  - All-NBA Second Team: 2017, 2018

Figure 7: Giannis Antetokounmpo.

We are interested in the NBA Regular season table. Therefore we need to navigate to this one. We want to extract for the row **Year 2019-20** and the columns points per game, blocks per game and rebounds per game.

For your statistics you are going to compare the best players from the teams in the conference semifinals. "Best" is defined by the highest count of points per game in our case.

The three best players of each team make it into our comparison pool[9] If statistics for a player is missing it is fine to ignore this player. Since you are only selecting the top 3 from each team you could just give these people the lowest possible score (this way they are not included).

We want to have a look at the best player with respect to points per game, with respect to blocks per game and rebounds per game.

Therefore we will plot the player over the points/blocks/rebounds. You can choose your favorite plotting tool for this. We would like to have players of each team grouped together (this can be color-coded or with a bracket, or another visualization).

Your implementation should be saved to a file called `fetch_player statistics.py`.

The produced plots should be saved.

Steps for creating the script

- visit `https://en.wikipedia.org/wiki/2020_NBA_playoffs`

[9] Since we take the 3 best players of each team, we will end up comparing the abilities of 24 players in total.

- crawl through the team and player wikipedia websites via the internal urls as described above
- define the best 3 players of each team in the conference semifinals based on the ”Points per game” in 2019/20
- fetch their statistics in the categories points per game, blocks per game and rebounds per game
- create three plots (players of each team grouped together in the plots):
  - Players over points per game
  - Players over blocks per game
  - Players over rebounds per game
- Save the plots!

Files to Deliver in this Subtask Create a folder `NBA_player_statistics` where you put all output files from our test runs created for solving this subtask.

Files Required in this Subtask

- `fetch_player statistics.py`
- `NBA_player_statistics/players_over_ppg.png`
- `NBA_player_statistics/players_over_bpg.png`
- `NBA_player_statistics/players_over_rpg.png`

5.6 Challenge - Wiki Race with URLs (5 bonus points and a price for the winner!)

Everyone has probably heard of golf. You try to get the ball into the hole with the least amount of hits. Let us play some Wikipedia golf.

Write a script using your previous functions that finds the shortest way (in number of urls to visit) from `https://en.wikipedia.org/wiki/Parque_18_de_marzo_de_1938` to `https://en.wikipedia.org/wiki/Bill_Mundell` using only urls in Wikipedia articles. The script should work with any wikipedia url.

The main objective is to find the shortest path. You can expect to work on the english wiki only for this task. The websites given will be your test case.

Note: Since the simplest approach might technically work, but not be the fastest finding the solution, we would like you to still be able to provide a solution for bonus points, therefore you can find the shortest path lengths between these two wikipedia articles: `https://en.wikipedia.org/wiki/Nobel_Prize` and `https://en.wikipedia.org/wiki/Array_data_structure`.

In order to assign a winner, we will evaluate this for 2 undisclosed urls. The most efficient (and correct) script will win a prize :)

Your chances for winning are even higher, if your script is fast.

Note: You are going to be on english wikipedia en.wiki .

You have to use python ;)

Files to Deliver in this Subtask

Create a folder `wiki_race_challenge` where you put all output files from our test runs created for solving this subtask.

Files Required in this Subtask

- `wiki_race_challenge`.py
- `wiki_race_challenge/shortest_way.txt`

Wohoo - You finished another assignment! Congrats!

Update October, 13th 2020

- check Section 5.2 - anchor explanation added, added requirements for urls, identifier example
- check Section 5.3 - added 3 Notes, defined date formats more precisely)

Update October, 14th 2020

- check Section 5.3 - defined that yyyy/mm[/dd] , where only the day optional

Update October, 15th 2020 Q&A

- **For subtask 5.3, are we meant to convert the date with regular expressions, or can we use a standard library?** - Regex should be used everywhere in task 5.3 and 5.3 - so yes, use regex please.
- **Do we need to use a regex to fetch the venue in 5.4?** - No need to fetch it with regex in 5.4

Update October, 16th 2020 Updated Task 5.2, 5.3 and 5.6 questions. **Q&A**

- **For subtask 5.3, are we meant to convert the date with regular expressions, or can we use a standard library?** - Regex should be used everywhere in task 5.3 and 5.3 - so yes, use regex please.
- **What's a relative url and what's a base url?** - The url you see when searching for Giannis Antetokounmpo in your browser is the full or absolute url `https://en.wikipedia.org/wiki/Giannis_Antetokounmpo`. An absolute URL is the entire address from the protocol (https), to the domain name including the location within your website (path). A relative url does not use the full web address but only contains the path followed by the domain. Therefore it assumes the link is on the same site and is part of the same root domain. The relative url linking to Giannis Antetokounmpos wiki page looks like this:

```
<a href=”/wiki/Giannis_Antetokounmpo”
```

The base url is represented by `https://en.wikipedia.org/` in our example.

- **Do all of the dates in 5.3 be formatted the same? As in either 06 or 6 for june, or is it fine that they are mixed?** The dates should al be formatted in the same style, which is **yyyy\mm\dd**, so june would be represented with 06.

- **(5.3) When sorting should we use regex for that as well or is it fine to use inbuilt functions?** For the sorting of the list you can use a built-in function.

- **In 5.3 it says we should get the dates without days as well. Im wondering if this is needed for ISO dates as well or only the other formats?** Yes, this is required for the iso dates as well. So basically all date types you include in the yyyymmdd version of the function, should be included.

Update October, 19th 2020 Updated 5.3 completely and simplified challenge. Updated Q& A. **Q&A**

- **5.3 - Should we remove duplicate dates?** - You can keep duplicate dates without us withdrawing points.

- **5.4 - In the video lecture about beautiful soup I saw that you were using the lxml parser. Are we allowed to use it, or do we need to use the standard python html.parser?** - You can always use what you learned in the lecture.

- **5.4 - Are we supposed to focus on the 19-20 or 20-21 season?** -Your implementation should work on both pages, and you just test both, to verify this. The 2019-20 table contains maybe a little bit more interesting information, so I would focus on making this one work first.

- **5.4 - Are we meant to use only soup or soup and regex in 5.4?** - 5.4 is a beautiful soup task , so you can use beautiful soup only :)

- **5.4 - In task 5.5 there's a lot of players with missing information for example:** `https://en.wikipedia.org/wiki/Johnathan_Motley` **misses the 2019-20 season** `https://en.wikipedia.org/wiki/Paul_Watson_(basketball)` **misses a stats table** - If statistics for a player is missing it is fine to ignore this player. Since you are only selecting the top 3 from each team I would say you could just give these people the lowest possible score (this way they are not included).