

Mandatory Assignment 5

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

University of Oslo - IN3110/IN4110

Fall 2021

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 and operating system) as well as how to install them. Documentation on how to run your examples 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 docstrings¹. However, you are free to choose your own docstring style. We expect your code to be well-formatted and readable. Black is useful tool for automatically formatting Python code². Coding style and documentation will be part of the point evaluation for **all tasks** in this assignment. This assignment will be followed by a peer-review. Delivery of the assignment is required for participation in the peer review.

A Note on Your Solutions There is no such thing as the “one right solution” to the assignments. As a response to the survey, we wanted to mention here that the assignment text is kept relatively open as we want to give you enough freedom to implement the solution in a way that you feel works best. In real software development, you will not have a step-by-step exact guide on how to implement a certain feature. You will likely know, what the product is supposed to look like in the end and will have to figure out a way, how to best create a product with certain features. With these assignments, we want to give you a chance on learning how to tackle a problem, how to structure a project and realizing that there is not necessarily one perfect solution.

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.

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

¹https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html

²<https://black.readthedocs.io>

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. Writing tests to run with `pytest` is a great way to check your code as you write it, and give us a clear idea of how you expect your code to work.
- `regex101.com` is a great resource for testing and understanding your regular expressions, as illustrated in our lecture on 6. October.

5.1 Sending HTTP 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 `requests` module ³.

The module can be installed as follows:

```
1 python3 -m 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.

```
1 #import the module
2 import requests as req
3
4 # grabbing the content of https://en.wikipedia.org/wiki/URL
5 resp = req.get("https://en.wikipedia.org/wiki/URL")
6
7 #get() method returns a response object
8 print(resp.text)
9
10 <!DOCTYPE html>
11 <html class="client-nojs" lang="en" dir="ltr">
12 <head>
13 <meta charset="UTF-8"/>
14 <title>URL - Wikipedia</title>
15 ...
16 #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?

³<https://requests.readthedocs.io/en/master/user/install/>

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. `https://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:

```
1 import requests as req
2
3 params = {'key1': 'value1', 'key2': 'value2'}
4 r = req.get('https://httpbin.org/get', params=params)
5
6 # the url would be encoded accordingly
7 print(r.url)
8 https://httpbin.org/get?key2=value2&key1=value1
```

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

```
1 import requests as req
2
3
4 params = {'user_name': 'admin', 'password': 'password'}
5 r = req.get('http://httpbin.org/get', params=params)
6
7 print(r.url)
8 http://httpbin.org/get?user_name=admin&password=password
9
10 # print the .text content of the request
11 print(r.text)
12
13 {
14     "args": {
15         "password": "password",
16         "user_name": "admin"
17     },
18     "headers": {
19         "Accept": "*/*",
20         "Accept-Encoding": "gzip, deflate",
21         "Host": "httpbin.org",
22         "User-Agent": "python-requests/2.22.0",
23         "X-Amzn-Trace-Id": "Root=1-5f841066-0c0ea5084573be8a230feaea"
24     },
25     "origin": "51.175.234.27",
26     "url": "http://httpbin.org/get?user_name=admin&password=password"
27 }
```

Your task will be to create the function `get_html` which makes a request for a url from a given website. The function should be able to optionally take in parameters that are 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 html text is returned.

An outline of the function is shown below.

```
1
```

```

2 # import what you need
3 import requests as req
4
5
6 def get_html(url, params=None, output=None):
7     """Your detailed Docstring Here
8     """
9
10    # passing the optional paramters argument to the get
    function
11    response = req.get(url, params=params)
12
13    #YOUR CODE
14    # if output is specified, the response txt and url get
    printed to a
15    # txt file with the name in 'output'
16
17    return html_str
18
19 html_str = get_html(url)
20 html_str = get_html(url, params={"key": "value"})
21 html_str = get_html(url, params={"key": "value"}, output="output.
    txt")
22
23 # if 'output' is specified, the response txt and url get printed
    to a file
24 # with the name given in output.
25 # For example:
26
27 url: https://en.wikipedia.org/w/index.php?key=value
28 <html>
29 ...
30 </html>

```

You should test your function on the following websites:

- https://en.wikipedia.org/wiki/Studio_Ghibli
- https://en.wikipedia.org/wiki/Star_Wars

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

Files to deliver in this Subtask

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

- `requesting_urls.py`
- Three files inside the `requesting_urls` folder containing the function output for the 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.

```
1 list_of_urls = find_urls(html_str)
```

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 anchor⁴ 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` after stripping.

In order to handle relative urls, the function should have an optional parameter to receive a base url.

What is a relative url and what is 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 should be retrieved from the same host. The relative url linking to Giannis Antetokounmpo's wiki page looks like this:

```
1 <a href="/wiki/Giannis_Antetokounmpo" ...
2
```

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

For getting started, here is an example test to verify finding links in an HTML snippet:

```
1 def test_find_urls():
2     html = """
3     <a href="#fragment-only">anchor link</a>
4     <a id="some-id" href="/relative/path#fragment">relative link</a>
5     <a href="//other.host/same-protocol">same-protocol link</a>
6     <a href="https://example.com">absolute URL</a>
7     """
8     urls = find_urls(html, base_url="https://en.wikipedia.org")
9     assert urls == [
10         "https://en.wikipedia.org/relative/path",
11         "https://other.host/same-protocol",
12         "https://example.com",
13     ]
```

⁴name of the HTML-element, with the HTML-tag "a"

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, by showing all linked URLs and all linked wikipedia pages for the following pages:

- `https://en.wikipedia.org/wiki/Nobel_Prize`
- `https://en.wikipedia.org/wiki/Bundesliga`
- `https://en.wikipedia.org/wiki/2019%E2%80%93FIS_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 deduct points for it).

Note 2: You should only find urls that are in the `href` attribute of 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 handle partial urls, which should 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 `find_urls` and `find_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. Feel free to create helper functions, if you feel that would improve your implementation.

Shown below is a possible general outline of the functions to be created in this task

```
1
2 def find_urls(html_string, base_url=<your choice of a default or
  None here>, output=None):
3     """Your Descriptive Docstring Here
4     """
5     # YOUR CODE HERE
6
7     # Remove duplicate urls
8
9     # write list of urls to file if output not None
10
11     return url_list
```

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

```

12
13
14 def find_articles(html_string, output=None):
15     """Your Descriptive Docstring Here
16     """
17     urls = find_urls(text)
18
19     # write list of articles found to file if output not None
20
21     return article_url_list

```

For reference, here is an example of using regex to find the `src` attribute of all `img` tags:

```

1 def find_img_src(html):
2     """Find all src attributes of img tags in an HTML string
3
4     Args:
5
6         html (str): A string containing some HTML.
7
8     Returns:
9
10        list: A list of strings
11
12        The list contains every found src attribute
13        of an img tag in the given HTML.
14    """
15    # img_pat finds all the  snippets
16    # this finds <img and collects everything up to the closing
17    # '>'
18    img_pat = re.compile(r"<img[^\>]+>", flags=re.IGNORECASE)
19    # src finds the text between quotes of the 'src' attribute
20    src_pat = re.compile(r"src=\"([^\"]+)\"")
21    src_list = []
22    # first, find all the img tags
23    for img_tag in img_pat.findall(html):
24        # then, find the src attribute of the img, if any
25        src = src_pat.find(html)
26        for src in src_pat.findall(html):
27            src_list.append(src)
28    return src_list

```

You will need to adapt this to find the `href` attribute of a tags, and handle the additional processing of evaluating URLs relative to the page.

Files to Deliver in this Subtask

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

Files Required in this Subtask

- `filter_urls.py` containing the `find_urls` and `find_articles` functions
- Six files inside the `filter_urls` folder containing a list of the urls and articles returned for each of the three example websites.

You should not use BeautifulSoup 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 year/month/day format:

```
1 1998/10/31
2 1998/11/04
3 1999/01/13
```

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

```
1 DMY: 13 October 2020
2 MDY: October 13, 2020
3 YMD: 2020 October 13
4 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.

```
1 DMY: 13 Oct 2020
2 MDY: Oct 13, 2020
3 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.

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

```
1 1998/10/31
2 1998/11/18
3 1999/01/13
```

The dates **returned** should be formatted as year/month/day.

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

You are supposed to use regex in this task. One strategy could 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 expression 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:

- visit the following websites
- https://en.wikipedia.org/wiki/J._K._Rowling

⁶<https://en.wikipedia.org/wiki/Template:Date>

- https://en.wikipedia.org/wiki/Richard_Feynman
- https://en.wikipedia.org/wiki/Hans_Rosling

Below you can find an example outline of the function

```

1
2 def find_dates(html_string, output=None):
3     """
4     Finds all dates in a html string.
5
6     The returned list is in the following format:
7     - 1998/10/12
8     - 1998/11/04
9     - 1999/01/13
10
11     The following formats are considered when searching:
12     DMY: 13 Oct(ober) 2020
13     MDY: Oct(ober) 13, 2020
14     YMD: 2020 Oct(ober) 13
15     ISO: 2020-10-13
16
17     # Finish Docstring Here
18
19     Returns:
20     results (list): A list with all the dates found in Y/M/D
21     format
22     """
23
24     return results

```

Feel free to create helper functions, if it helps you organize your code. Steps that you likely will need to cover within this task are the following

- get html
- use `re.findall` to extract a list of dates in the formats DMY, MDY, YMD, ISO
- convert the date with `re.sub`

Feel free to divide your regular expressions into small building blocks containing a day, month, and year part. The month december could be a building block represented like this matching dec or december with optional capitalization.

```

1 dec = r"\b[dd]ec(?:ember)?\b"

```

To check for the month, you might want to define all the months in such a pattern and match these like this for december and november

```

1 months = rf"(?:{nov}|{dec})"

```

You also have to consider matching the ISO format, where months are represented as digits Take into account that months represented by a single digit have a leading zero, e.g. 06 for june.

```
1 iso_month_format = r"\b(?:0\d|1[0-2])\b"
```

Using the building blocks you created you can find the DMY format for example using the following expression:

```
1 # defining DMY format
2 dmy = rf"{day}\s{months}\s{year}"
3
4 # finding all dmy format matches using re.findall
5 re.findall(rf"{dmy}", html_text)
```

This may not be the most efficient way nor the only possible solution, but it might help defining the whole regular expression you are about to create.

For the "year" you can assume that years will have at least 4 digits, e.g. 1000 and later.

For converting the date with regular expressions from DMY, MDY, YMD, ISO to YYYY/MM/DD, you can split the conversion into two steps. Assuming you have a list of dates you extracted you can first reorder all formats using `regub`. This could look like this for the DMY format:

```
1 # reformat DMY as Y/M/D
2 date_element = re.sub(rf"({day})\s({month})\s({year})", r"\3/\2/\1",
3                        date_element)
```

In the second step you can replace the months names by the number that represents that month.

One option to replace the string for the month with the digit would be using the `startswith()` method. An example snippet is shown below.

```
1 # replace december string with digits
2 if month.lower().startswith("dec"):
3     # Your Code
```

Save your implementation to `collect_dates.py`.

Files to Deliver in this Subtask

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

- `collect_dates.py` containing the function `find_dates`
- Three files inside the `collect_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 finding dates (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)

You can install Beautiful Soup via:

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`.

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

[illegible]

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.

```

1 from bs4 import BeautifulSoup
2 import requests as req
3
4 url = "https://en.wikipedia.org/wiki/List_of_soups"
5
6 # request url as we have already learned - yeah!
7 request = req.get(url)
8
9 soup = BeautifulSoup(request.text, "html.parser")
10
11 #check the title of the wikipedia page
12 print(soup.title)
13
14 # get the soup table
15 soup_table = soup.find('table', {"class": 'wikitable sortable'})

```

Now we need to know how to navigate to the table. To figure out which column holds which information, we can look at the header. We will use the html `<th>` tag (short for “table header”), which defines the header cell in an html table. An html table has two 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/2020%E2%80%9321_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 for either Women or Men (see figure below).

#	Event	Date	Venue	Type	Details
1782	1	16 October 2020	 Solitude	GS 405	
1784	2	16 November 2020	 Lech/Zoo	PS 401	
		28 November 2020	 Lake Louise	GS 404	
		4 December 2020	 Beaver Creek	GS 404	
		6 December 2020	 Beaver Creek	GS 404	
1786	3	6 December 2020	 Val d'Isère	GS 401	
1788	4	6 December 2020	 Val d'Isère	GS 401	
1787	5	10 December 2020	 Val d'Isère	GS 401	
1788	6	10 December 2020	 Val d'Isère	GS 401	
1789	7	10 December 2020	 Val d'Isère	GS 401	
1790	8	10 December 2020	 Val Gardena/Schöden	GS 401	
1791	9	20 December 2020	 Val Gardena/Schöden	GS 401	
1792	10	20 December 2020	 Val Gardena/Schöden	GS 401	
1793	11	20 December 2020	 Val Gardena/Schöden	GS 401	
1794	12	20 December 2020	 Val Gardena/Schöden	GS 401	
1795	13	20 December 2020	 Val Gardena/Schöden	GS 401	
1796	14	20 December 2020	 Val Gardena/Schöden	GS 401	
1797	15	20 December 2020	 Val Gardena/Schöden	GS 401	
1798	16	20 December 2020	 Val Gardena/Schöden	GS 401	
1799	17	20 December 2020	 Val Gardena/Schöden	GS 401	
1800	18	20 December 2020	 Val Gardena/Schöden	GS 401	
1801	19	20 December 2020	 Val Gardena/Schöden	GS 401	
1802	20	20 December 2020	 Val Gardena/Schöden	GS 401	
1803	21	20 December 2020	 Val Gardena/Schöden	GS 401	
1804	22	20 December 2020	 Val Gardena/Schöden	GS 401	
1805	23	20 December 2020	 Val Gardena/Schöden	GS 401	
1806	24	20 December 2020	 Val Gardena/Schöden	GS 401	
1807	25	20 December 2020	 Val Gardena/Schöden	GS 401	
1808	26	20 December 2020	 Val Gardena/Schöden	GS 401	
1809	27	20 December 2020	 Val Gardena/Schöden	GS 401	
1810	28	20 December 2020	 Val Gardena/Schöden	GS 401	
1811	29	27 February 2021	 Cortina	GS 401	
1812	30	28 February 2021	 Cortina	GS 401	
1813	31	6 March 2021	 Cortina	GS 401	
1814	32	7 March 2021	 Cortina	GS 401	
1815	33	13 March 2021	 Cortina	GS 401	
1816	34	14 March 2021	 Cortina	GS 401	
1817	35	17 March 2021	 Cortina	GS 401	
1818	36	18 March 2021	 Cortina	GS 401	

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`.

Below you can find some pseudo-code. This code is not finished, but can give you an inspiration how to set up your implementation of this task.

```
1
2 import re
3 from bs4 import BeautifulSoup
4
5
6 def extract_events(url):
7     """Extract date, venue and discipline for competitions.
8     Your documentation here.
9     Args:
10         url (str): The url to extract events from.
11     Returns:
12         table_info (list of lists): A nested list where the rows
13         represent each
14         race date, and the columns are [date, venue,
15         discipline].
16     """
17
18     disciplines = {
19         "DH": "Downhill",
20         "SL": "Slalom",
21         "GS": "Giant Slalom",
22         "SG": "Super Giant Slalom",
23         "AC": "Alpine Combined",
24         "PG": "Parallel Giant Slalom",
25     }
26
27     # get the html
28     html = get_html(url)
29
30     # make soup
31     soup = BeautifulSoup(html, "html.parser")
32
33     # Find the tag that contains the Calendar header span
34     calendar_header = soup.find(id="Calendar")
35
36     # Find the following table
37     calendar_table = calendar_header.find_all_next("table")[0]
38
39     # Find the rows of the first table
40     rows = calendar_table.find_all("tr")
41
42     # try parsing the row of 'th' cells to identify the indices
43     # for Event, Venue, and Type (discipline)
44
45     found_event = None
46     found_venue = None
47     found_discipline = None
48     # Saving all necessary values in the list under
49     events = []
50
51     # how many columns does a full row have?
52     full_row_length = 10 # This is wrong! Can you find the right
53     value?
54     # some rows have fewer because the 'venue'
55     # spans multiple rows,
```

```

53 # short_row_length means a repeated venue, which should be re-
54 used
55 # from the previous iteration
56 short_row_length = full_row_length - 2
57
58 for row in rows:
59     cells = row.find_all("td")
60
61     # begin debug: to help figure out what to write,
62     # show the contents of each row
63     print(f"new row: {len(cells)} cells")
64     for idx, cell in enumerate(cells):
65         print(f"  cell {idx}: {cell}")
66     # end debug: remove when it works!
67
68     # some rows have one number of columns,
69     # if it's a different number (usually 0 or 1),
70     # ignore it
71     if len(cells) not in {full_row_length, short_row_length}:
72         # skip rows that don't have most columns
73         continue
74     # Hard-coding indexes works for now (you should find them
75     # using the Header!)
76     event = cells[1]
77     # An event seems to always be a 1-3 digit number, so we
78     # can check that we have an event with a simple regex
79     if re.match(r"\d{1,3}", event.text.strip()):
80         found_event = event.text.strip()
81     else:
82         found_event = None
83
84     if len(cells) == full_row_length:
85         # If event is cancelled, the index below might need to
86         # be shifted.
87         venue_cell = cells[3]
88         found_venue = venue_cell.text.strip()
89         discipline_index = 5
90     else:
91         # repeated venue, discipline is in a different column
92         # where is the discipline column?
93         discipline_index = ?
94
95     discipline = cells[discipline_index]
96     # find the discipline id
97     # can you make a regex to find only the keys of the
98     # disciplines dictionary?
99     # (DH|...)
100     discipline_regex = YOUR_CODE_HERE
101     # this can also be done with just HTML parsing
102     discipline_match = re.search(discipline_regex, discipline.
103     text.strip())
104     if discipline_match:
105         # look up the full discipline name
106         found_discipline = YOUR_CODE_HERE
107     else:
108         found_discipline = None

```

```

104         if found_venue and found_event and found_discipline:
105             # if we found something
106                 events.append((found_event, found_venue,
107                               found_discipline))
108     return events

```

Since you would like to gamble with your friends beforehand you will create a betting slip. That slip will contain 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 using Markdown ⁷ or HTML formatting. If you choose Markdown, <https://markdownlivepreview.com> is a handy tool for previewing your markdown rendering, to make sure it's correct.

Here is an example of a function you could use to create the betting slip in Markdown format.

```

1 def create_betting_slip(events, save_as):
2     """Saves a markdown format betting slip to the location
3         './datetime_filter/<save_as>.md'.
4         Args:
5             events (list): takes a list of 3-tuples containing
6                             date, venue and
7                             type for each event.
8             save_as (string): filename to save the markdown
9                             betting slip as.
10            """
11            # ensure directory exists
12            os.makedirs("datetime_filter", exist_ok=True)
13
14            with open(f"./datetime_filter/{save_as}.md", "w") as out_file:
15                out_file.write(f"# BETTING SLIP ({save_as})\n\nName:\n\n")
16                out_file.write("Date | Venue | Discipline | Who wins?\n")
17                out_file.write(" --- | --- | --- | --- \n")
18                for e in events:
19                    date, venue, type = e
20                    out_file.write(f"{date} | {venue} | {type} | \n")

```

Files to Deliver in this Subtask

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.

An example betting slip could look like this:

⁷<https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables>

BETTING SLIP
Name:

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

Figure 3: Betting Slip.

5.5 NBA Player Statistics Season 2020/2021 (IN3110 & IN4110 15 Points)

Go Milwaukee Bucks! Let's imagine you are into basketball. After this unpredictable season, you really want to dive into the player statistics. Lucky you! In this task you are going to write a script `fetch_player_statistics.py` which visits the "2021 NBA playoffs" website on wikipedia (this one: https://en.wikipedia.org/wiki/2021_NBA_playoffs) and creates some player statistics.

This task can be broken down into three subtasks:

1. Get the list of teams.
2. Given a team url, get the list of players.
3. Given a player url, get their statistics.

As in task 5.4 we will use `BeautifulSoup` for parsing.

5.5.1 Get the List of Teams.

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.

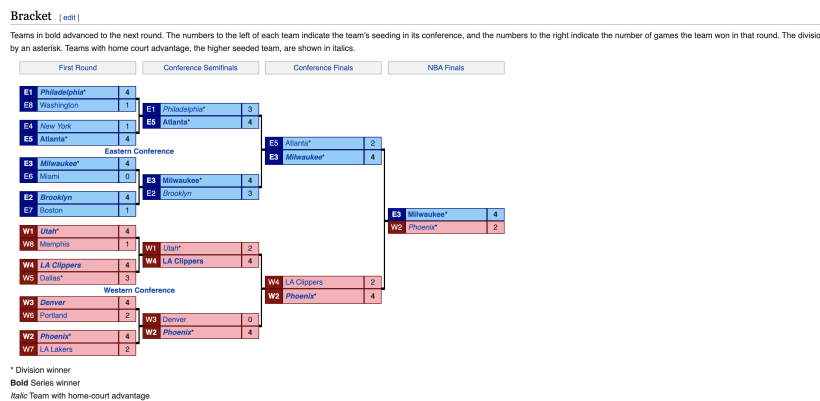


Figure 4: Bracket Section.

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.

5.5.2 Get the List of Players, given a Team URL.

You will follow the urls to the wikipedia websites of the corresponding teams in the Bracket section.

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

⁸Hint: It should be 8.

Article	Talk	Read	Edit	View history	Search Wikipedia
---------	------	------	------	--------------	------------------

<div> <div>2020–21 Milwaukee Bucks season</div> <div>From Wikipedia, the free encyclopedia</div> </div> <div> <p>The 2020–21 Milwaukee Bucks season was the 53rd season of the franchise in the National Basketball Association (NBA). On April 30, 2021, the Bucks clinched the Central Division for a record 10th time with a win over the Chicago Bulls. The Bucks finished the season with a 46–26 record, good enough for the third seed. The Bucks began their playoff run by sweeping the sixth seeded Miami Heat in the opening round, in a rematch of Eastern Conference Semi-finals the previous year. The Bucks then defeated the second seeded Brooklyn Nets in the Eastern Conference Semi-finals in seven games. In the Eastern Conference Finals, the Bucks defeated the Atlanta Hawks in six games, reaching the NBA Finals for the first time in 47 years, and winning the Eastern Conference Finals for the first time in franchise history^[1] The Bucks are the first NBA team to have won both a Western Conference and Eastern Conference championship in their history, as they were in the Western Conference when they reached the NBA Finals in 1971 and 1974. The Bucks would face the Phoenix Suns in the NBA Finals. Despite losing the first two games, the Bucks won four straight, winning 4–2 and clinching their second NBA title, the first since 1971. This was seen as even more of a win considering the running name of "Bucks in 6"^[2]</p> </div> <div> <div>Contents [hide]</div> <div> <div>1</div> <div>Draft picks</div> </div> <div> <div>2</div> <div>Roster</div> </div> <div> <div>3</div> <div>Standings</div> </div> <div> <div>3.1</div> <div>Division</div> </div> <div> <div>3.2</div> <div>Conference</div> </div> <div> <div>4</div> <div>Game log</div> </div> <div> <div>4.1</div> <div>Preseason</div> </div> <div> <div>4.2</div> <div>Regular season</div> </div> <div> <div>4.3</div> <div>Playoffs</div> </div> <div> <div>5</div> <div>Transactions</div> </div> <div> <div>5.1</div> <div>Trades</div> </div> <div> <div>5.1.1</div> <div>Failed Bogdan (Bogdanovsk) trade</div> </div> <div> <div>5.2</div> <div>Free agency</div> </div> <div> <div>5.2.1</div> <div>Re-signed</div> </div> <div> <div>5.2.2</div> <div>Additions</div> </div> <div> <div>5.2.3</div> <div>Subtractions</div> </div> <div> <div>6</div> <div>References</div> </div> </div>
--

Draft picks

[edit]

Main article: 2020 NBA draft

Round	Pick	Player	Position(s)	Nationality	School
1	24	R. J. Hampton	PG/SG	 United States	New Zealand Breakers
2	45	Jordan Nwora	SF	 Nigeria	Louisville

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. In your script this could be done by a function `extract_player` which will allow you to extract the player urls and names based on the team url. The url to each player can be found in the roster. Therefore you need to navigate to the table holding the team roster. For the Milwaukee Bucks the roster section looks like this:

Roster

Figure 6: Roster Milwaukee Bucks.

5.5.3 Given a Player URL, get Their Statistics.

Fantastic, let's start on the player statistic section! In order to filter the statistics for one specific player you need the output the player name and url to this players wikipedia page, which you extracted in the last step. The player url can be used as an input for the function `extract_player_statistics` which you will create in this subtask.

You will request the url to that player. If we follow the link to Giannis Antetokounmpo the figure below shows the part of the webpage showing the player statistics.

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
2020-21†	Milwaukee	61	61	33.0	.569	.303	.685	11.0	5.9	1.2	1.2	28.1
Career		589	524	32.5	.532	.287	.717	9.1	4.5	1.2	1.3	20.9
All-Star		4	4	26.8	.653	.231	.667	8.8	3.0	1.3	1.0	27.3

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 2020-21** 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. Since we take the 3 best players of each team, we will end up comparing the abilities of **24 players in total**.

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 compare these selected players with respect to points per game, 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.
An example plot is shown below.

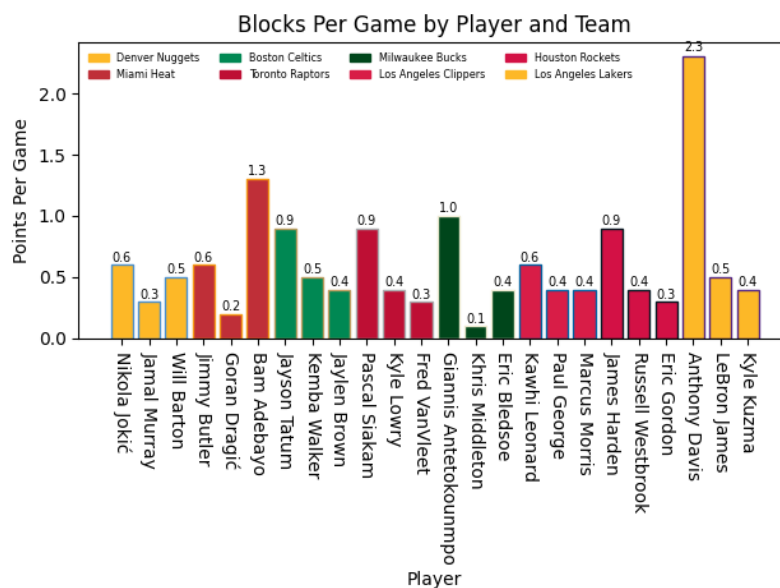


Figure 8: Example Plot.

Steps for creating the script

- visit https://en.wikipedia.org/wiki/2021_NBA_playoffs
- 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 2020/2021
- 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!

You are free to structure your script the way you think is best. However, we have outlined below an example of how you could structure your implementation:

```
1 from bs4 import BeautifulSoup
2
3 from requesting_urls import get_html
4
5
6 def extract_teams():
7     """Extract team names and urls from the NBA Playoff 'Bracket'
8     section table.
9
10    Returns:
11        team_names (list): A list of team names that made it to
12        the conference
13        semifinals.
14        team_urls (list): A list of absolute Wikipedia urls
15        corresponding to team_names.
16
17    """
18
19    # get html using for example get_html from requesting_urls
20    html = get_html(url)
21
22    # create soup
23    soup = BeautifulSoup(html, "html.parser")
24    # find bracket we are interested in
25    bracket_header = soup.find(id="Bracket")
26    bracket_table = bracket_header.find_next("table")
27    rows = bracket_table.find_all("tr")
28
29    # create list of teams
30    team_list = []
31
32    for i in range(1, len(rows)):
33        cells = rows[i].find_all("td")
34        cells_text = [cell.get_text(strip=True) for cell in cells]
35
36        # Filter out the cells that are empty
37        cells_text = [cell for cell in cells_text if cell]
38
39        # Find the rows that contain seeding, team name and games
40        won
41        if len(cells_text) > 1:
42            YOUR_CODE
43
44    # Filter out the teams that appear more than once, which means
45    they made it
46    # to the conference semifinals
47    team_list_filtered = YOUR_CODE
48
49    # create lists of team names and urls to the team website
50    team_names = []
51    team_urls = []
52
53    # your code
```

```

49     return team_names, team_urls
50
51
52
53 def extract_players(team_url):
54     """Extract players that played for a specific team in the NBA
55     playoffs.
56
57     Args:
58         team_url (str): URL to the Wikipedia article of the season
59         of a given
60         team.
61
62     Returns:
63         player_names (list): A list of players names corresponding
64         to the team whos URL was passed.
65         semifinals.
66         player_urls (list): A list of Wikipedia URLs corresponding
67         to
68         player_names of the team whos URL was passed.
69
70     """
71
72     # keep base url
73     base_url = "https://en.wikipedia.org"
74
75     # get html for each page using the team url you extracted
76     before
77     html = get_html(team_url)
78
79     # make soup
80     soup = BeautifulSoup(html, "html.parser")
81     # get the header of the Roster
82     roster_header = soup.find(id="Roster")
83     # identify table
84     roster_table = roster_header.find_next("table")
85     rows = roster_table.find_all("tr")
86
87     # prepare lists for player names and urls
88     player_names = []
89     player_urls = []
90
91     for i in range(0, len(rows)):
92         cells = rows[i].find_all("td")
93         cells_text = [cell.get_text(strip=True) for cell in cells]
94
95         if len(cells_text) == 7:
96             rel_url = cells[2].find_next("a").attrs["href"]
97             # Use e.g. regex to remove information in parenthesis
98             following the name
99             YOUR_CODE
100             # create urls to each player
101             # need to create absolute urls combining the base and
102             the relative url
103             player_urls.append(base_url + rel_url)
104
105     return player_names, player_urls

```

```

99
100
101 def extract_player_statistics(player_url):
102     """Extract player statistics for NBA player.
103
104     # Note: Make yourself familiar with the 2020-2021 player
105     statistics wikipedia page and adapt the code accordingly.
106
107     Args:
108         player_url (str): URL to the Wikipedia article of a player
109         .
110
111     Returns:
112         ppg (float): Points per Game.
113         bpg (float): Blocks per Game.
114         rpg (float): Rebounds per Game.
115
116     """
117     # As some players have incomplete statistics/information, you
118     can set a default score, if you want.
119
120     ppg = 0.0
121     bpg = 0.0
122     rpg = 0.0
123
124     # get html
125     html = get_html(player_url)
126
127     # make soup
128     soup = BeautifulSoup(html.text, "html.parser")
129
130     # find header of NBA career statistics
131     nba_header = soup.find(id="NBA_career_statistics")
132
133     # check for alternative name of header
134     if nba_header is None:
135         nba_header = soup.find(id="NBA")
136
137     try:
138         # find regular season header
139         # You might want to check for different spellings, e.g.
140         capitalization
141         # You also want to take into account the different orders
142         of header and table
143         regular_season_header = nba_header.find_next(id="
144         Regular_season")
145
146         # next we should identify the table
147         nba_table = regular_season_header.find_next("table")
148
149     except:
150         try:
151             # table might be right after NBA career statistics
152             header
153             nba_table = nba_header.find_next("table")
154
155         except:

```

```

149         return ppg, bpg, rpg
150
151     # find nba table header and extract rows
152     table_header = nba_table.find_all("th")
153     # YOUR CODE
154
155     # find the columns for the different categories
156     ppg_column = YOUR_CODE_HERE
157     # YOUR CODE HERE
158
159     # Extract the scores from the different categories
160     scores = YOUR_CODE
161
162     # Convert the scores extracted to floats
163     # Note: In some cases the scores are not defined but only
164     # shown as '-'. In such cases you can just set the score to zero
165     # or not defined.
166     try:
167         scores[i] = float(scores[i])
168     except ValueError:
169         scores[i] = 0.0
170
171     return ppg, bpg, rpg

```

There are lots of ways to do the plotting, but here is one example using matplotlib, assuming one possible way to store your collected player data:

- `teams` is a dictionary with team names as keys and player lists as values
- each player in the list is a dictionary of player attributes, such as `name` or `ppg`

```

1 import matplotlib.pyplot as plt
2
3 teams = {
4     "Uio": [
5         {
6             "name": "Lisa",
7             "ppg": 10.0,
8         },
9         {
10            "name": "Vegard",
11            "ppg": 9.2,
12        },
13    ],
14    "Simula": [
15        {
16            "name": "Ingeborg",
17            "ppg": 11.2,
18        },
19        {
20            "name": "Min",
21            "ppg": 4.5,
22        },
23    ],
24 }

```



```

25
26 # a matplotlib color for each team name (could be a name or a #
    rrggbb web color string)
27 color_table = {
28     "UiO": "red",
29     "Simula": "orange",
30 }
31
32
33 def plot_NBA_player_statistics(teams):
34     """Plot NBA player statistics. In this case, just PPG"""
35     count_so_far = 0
36     all_names = []
37
38     # iterate through each team and the
39     for team, players in teams.items():
40         # pick the color for the team, from the table above
41         color = color_table[team]
42         # collect the ppg and name of each player on the team
43         # you'll want to repeat with other stats as well
44         ppg = []
45         names = []
46         for player in players:
47             names.append(player["name"])
48             ppg.append(player["ppg"])
49         # record all the names, for use later in x label
50         all_names.extend(names)
51
52         # the position of bars is shifted by the number of players
    so far
53         x = range(count_so_far, count_so_far + len(players))
54         count_so_far += len(players)
55         # make bars for this team's players ppg,
56         # with the team name as the label
57         bars = plt.bar(x, ppg, color=color, label=team)
58         # add the value as text on the bars
59         plt.bar_label(bars)
60
61         # use the names, rotated 90 degrees as the labels for the bars
62         plt.xticks(range(len(all_names)), all_names, rotation=90)
63         # add the legend with the colors for each team
64         plt.legend(loc=0)
65         # turn off gridlines
66         plt.grid(False)
67         # set the title
68         plt.title("points per game")
69         # save the figure to a file
70         plt.savefig("ppg.png")
71
72
73 plot_NBA_player_statistics(teams)

```

Files to Deliver in this Subtask Create a folder `NBA_player_statistics` where you store all the plots created in this task.

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 prize 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.wikipedia.org.

You have to use python ;)

Files to Deliver in this Subtask

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

Files Required in this Subtask

- `wiki_race_challenge.py`
- `wiki_race_challenge/shortest_way.txt` containing the list of URLs along the shortest path

You finished another assignment! Congrats!