Assignment 4: Web scraping

Contents

- Practical information
- Sending HTTP requests
- Regular Expressions for filtering URLs
- Regular Expressions for finding Dates **(IN4110 only)**
- Making your life easier with Soup, let's find something to celebrate!
- Preparing for summer Olympic Games in Paris 2024
- Before you submit
- Challenge - Wiki Race with URLs **(5 bonus points)**
- Turning it in

In this assignment you will be using `python`, regular expressions (`re`) and `beautifulsoup` for data mining. You will learn how to get the html code that makes up a website, parse through it for relevant information, and display the results. At the end there is a bonus task for playing wikipedia golf.

Updates:

- Fix medal examples for task 4.3
- Task 1.3: Suggest adding `base_url` to find_articles, to pass along to `find_urls`. It is not required, and if excluded, should behave the same as `base_url="https://en.wikipedia.org"` (i.e. calling `find_url` without specifying `base_url`).

Practical information

Total points (IN3110): 30 (+ 5 bonus points)

Total points (IN4110): 40 (+ 5 bonus points)

The assignment contains five subsets of tasks, in addition to a bonus challenge. Each subset

Skip to main content

skill within html parsing and regex.

Before you start, please note the following:

- Your solution to this mandatory assignment needs to be placed in the directory `assignment4` in your GitHub repository on the IN3110 org.
- We have a code skeleton for you containing the functions you need to write and corresponding tests. Make sure to keep the function names, arguments, and return types as specified as this makes grading easier. *However, you are free to write whatever helper functions you want.*
- Download the skeleton here. It should create the `assignment4` directory in your repo.
- Writing tests to run with `pytest` is not just a great way to check your code, but it also shows how you expect your code to work. Writing your own tests is greatly appreciated by the graders as it makes it much easier to grade your assignment.
- We have provided a collection of tests (in the `tests` directory in the code skeleton) representing a subset of the requirements for the assignment. All of these tests should **pass unmodified** if the assignment is complete and correct. To make it easier for you to test during code development, we have created certain test markers, which select specific functions to test corresponding to *one* task. To see all the markers type `pytest --markers` in your terminal. For instance, if you only want to run tests for task 1.1, type `pytest -m task11 tests/`.

A last tip on html and regex:

- In any task that asks you to parse some html code, make sure to inspect the webpage to see how this code is built up before proceeding. Here is a quick video guide on how to inspect the building blocks of a webpage.
- As you will soon find out, regex can be extremely powerful, but the syntax can be difficult to read and debug. We recommend regex101.com as a great resource for testing and understanding your regular expressions. Remember to document (with comments) how the regex is expected to work.

Sending HTTP requests

▶ Background on HTTP requests and making them with requests

> **Task 1.1: (2 points)** Complete the function `get_html` in `requesting_urls.py` which makes a request for a url from a given website.

Skip to main content

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 should be returned.

You should implement `get_html` so that it be called in the following ways:

```
html_str = get_html(url)
html_str = get_html(url, params={"key": "value"})
html_str = get_html(url, params={"key": "value"}, output="output.txt")
```

When `output` is specified, the file should contain the URL on the first line, followed by the response body:

```
https://example.com
<!doctype html>
...
```

You can test your implementation by running:

```
pytest -v -m task11 tests
```

Regular Expressions for filtering URLs

Next, you will be making functions for finding urls to other pages in a body of html using regex:

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

You will be using the `re.findall` method taught in the lecture.

> **Task 1.2 (3 points)** Complete the file named `filter_urls.py` which includes a function `find_urls` that receives an html string and returns a list of all urls found in the text.

You can test your implementation by running:

Skip to main content

To help you get started we have added a function `find_img_src(html: str)` to `filter_urls.py` that uses regex to find the `src` attribute of all `img` tags. You will need to adapt this to find the `href` attribute of `a` tags, and handle the additional processing of evaluating URLs that are paths on the same host (these start with `/`).

Note that:

- We only consider urls that are anchor hyperlinks with the `<a>` HTML tag.
- In order to handle url paths, the function should have an optional parameter to receive a base_url.
- Links to internal fragments to the current document (urls that start with `#`) should be excluded.
- The function should ignore fragment identifiers (link targets that begin with `#`). i.e., if the url points to a specific fragment/section using the hash symbol, the fragment part (the part starting with `#`) of the url should be stripped before it is returned by the function.

You should not use Beautiful Soup or any HTML parser in this task, only regular expressions.

Example of fragment: The string `https://www.example.com/somepage#someidentifier` becomes `https://www.example.com/somepage` after stripping the fragment.

What is a url path and what is a base url? Consider the English wikipedia page for Follobanen, for which the full url is https://en.wikipedia.org/wiki/Follo_Line. A *full* URL is the entire address, including the protocol (https), hostname (en.wikipedia.org), and path (`/wiki/Follo_Line`). A *path* URL is only the last part (starting with '/'), and does not include the protocol or hostname. As an example, Wikipedia pages (with base url https://en.wikipedia.org) will have the following path link to the Follo Line wiki page:

```
<a href="/wiki/Follo_Line">...</a>
```

> **Task 1.3 (1 point)** Further, make a function `find_articles` that calls `find_urls` and returns only urls to Wikipedia articles. This function should also use regex and **accept links to articles in any language**, e.g. (no.wikipedia.org, en.wikipedia.org, etc.).

Specifications:

- You should only return normal Wikipedia articles, so no special namespace article (or files). While you could exclude just the specific reserved namespaces, **it is sufficient to**

Skip to main content

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.

- 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.
- The found urls should be returned as a `set`.
- If an optional `output` filename argument is specified, the URLs should be written to this file, one URL per line.
- [optional] `base_url`, specifying the host where the HTML came from (same as `find_urls`). Default: `"https://en.wikipedia.org"`.

You can test your implementation by running:

```
pytest -v -m task13 tests
```

You should not use Beautiful Soup or any HTML parser in this task, only regular expressions.

Regular Expressions for finding Dates (IN4110 only)

In this task, you will be making functions for finding dates in a body of html using regular expressions.

> **Task 2 (10 points, IN4110 only)** Create a script named `collect_dates.py` that includes a function `find_dates` that receives an html string and returns a list of all dates found in the text in the following year/month/day format:

```
1998/10/31
1998/11/04
1999/01/13
```

The `collect_dates.py` provided has a skeleton of the `find_dates` function, and `tests/test_collect_dates` has some tests that should be able to pass if you have completed the task. You can check this by running

Skip to main content

```
pytest -v -m task2 tests
```

Specifications:

- You only need to consider the wikipedia standardized formats for dates. These are the four formats:

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

- The dates **returned** should be formatted as `year/month/day`. Month and day should be zero-padded, e.g. producing `2001/01/01`, not `2001/1/1`. The returned dates should be a *list*, matching the order of occurrence in the document.
- There should be an optional argument `output` allowing to specify that the resulting list will be saved to a text file with the name given, e.g.

```
find_dates("<html....", output="dates.txt").
```

Use regular expressions in this task. One strategy could be using the `re.findall` and `re.sub` methods. This will give us a list of all the strings matching your regular expression and allow you to change the formatting. But you are free to implement how you see fit, as long as you use regular expressions.

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` or another method.

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.

```
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

Skip to main content

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

You might find python's `"str".join(name_list)` function convenient for not having to repeat so much code. The function turns the elements in `name_list` into one string, with the string `"str"` in between.

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.

```
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:

```
# defining DMY format
dmy = rf"{day}\s{months}\s{year}"

# finding all dmy format matches using re.findall
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 (or expressions) you are about to create. You **do not** need to express all the possible formats in a single expression.

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 `re.sub`. One example for the DMY format:

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

Skip to main content

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

If you find yourself iterating over the whole document multiple times, it may be useful to track `match.start()`, which is the location in the document of the match as an integer in order to return a properly sorted list of dates.

Here's an example of sorting strings by an associated index, after storing them in tuples:

```
from operator import itemgetter

unsorted_list = [(100, "second"), (10, "first")]
# sort list by the first element in each tuple
sorted_list = sorted(unsorted_list, key=itemgetter(0))
# [(10, "first"), (100, "second")]
# extract items after sorting
sorted_items = [item[1] for item in sorted_list]
# sorted_items = ["first", "second"]
```

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

Making your life easier with Soup, let's find something to celebrate!

Want to have more reasons to celebrate with your friends? Say no more, we are going to dive deep into the anniversaries of the whole world and find many good reasons to make (almost) each day into a party. Did you know that July 17th is the World Emoji Day? Or that May 4th is Star Wars day? What about trying out some yoga at June 21st with your friends, which is a national day of yoga?

In this task, you will use the Python package BeautifulSoup4 to parse HTML. You are going to work with the wikipedia pages in the `Wikipedia:Selected_anniversaries` namespace, like this one for May. Here, you will fetch the *paragraphs* of the type:

May 1: **Beltane** and **Samhain** in the Northern and Southern Hemispheres, respectively; **Maharashtra Day** in Maharashtra, India (1960); **Loyalty Day** in the United States

which highlight some important anniversaries attributed to this day. You will focus on these paragraphs only, which play the role of a headline for the bullet list, and not on the contents of the bullet list (if you inspect the html body, the bullet list is an unordered list

Skip to main content

everything in front of the column `:`, and the corresponding events, which follow after the column, and are separated by the semicolon `;`. This will give you more practice with the structure of the html text and an opportunity to exercise more regex.

This task can be broken down into three subtasks:

1. Given an anniversaries article like May, extract the plain text from all of the paragraphs containing the highlighted anniversaries.
2. Given a list of strings of the format `'{Month} {day}: Event 1 (maybe some parenthesis); Event 2; Event 3, something, something\n'`, split each string into parts before and after ': ', further splitting the event part into separate events, if there are several. Transform the results into a dataframe with columns "Date" and "Event".
3. Given the dataframe, convert and store it as a markdown table.

You will be working inside the `find_anniversaries.py` script and you can test your implementation as you go with the `test_find_anniversaries.py` script using pytest. We have defined task specific markers for this task as well, so you can test your implementation as you go by, for instance, typing `pytest -v -m task31 tests`

Tip: It will really help for this task if you take a few minutes to familiarize yourself with the HTML structure of these wiki pages. We recommend you navigate to (for example) May, right click and select 'Inspect element'. Alternatively, right click on the web page and view the page source. Can you find the relevant paragraphs, like the one below, containing the highlighted anniversaries?

```
▼ <p>
  ▼ <b>
      <a href="/wiki/May_1" title="May 1">May 1</a>
    </b>
    ": "
  ▶ <b>…</b>
    " and "
  ▶ <b>…</b>
    " in the Northern and Southern Hemispheres, respectively; "
  ▶ <b>…</b>
    " in "
    <a href="/wiki/Maharashtra" title="Maharashtra">Maharashtra</a>
    ", India ("
    <a href="/wiki/1960" title="1960">1960</a>
    "); "
  ▶ <b>…</b>
    " in the United States "
  </p>
```

Hint: Search for `May 1` or `href="/wiki/May_1"`. Navigate to the place where this tag or text is encapsulated by a paragraph, i.e. it must exist within `<p> ... </p>`.

Getting ready to use Beautiful Soup

Skip to main content

You can install Beautiful Soup via:

```
python3 -m pip install beautifulsoup4
```

We would like to extract the paragraphs from the pages. First, we need to grab the HTML from the web page, so that we have something to parse through. Here is an example:

```
from bs4 import BeautifulSoup
import requests

url = "https://no.wikipedia.org/wiki/Aasta_Hansteen"

response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

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

# find all paragraphs
paragraphs = soup.find_all("p")
```

Given a wiki article, extract all highlighted anniversaries

> **Task 3.1 (3 points)** Inside `find_anniversaries.py`, complete the `extract_anniversaries` function, which should take an html text and a string holding the month, and return a list of strings containing the anniversaries.
>
> `extract_anniversaries` should:
>
> - extract all the paragraphs from the html body
> - filter the paragraphs down to the ones containing the highlighted anniversaries
> - extract the plain text from these paragraphs and store it as a string, one for each paragraph

Hint: There are usually a lot of paragraphs in an article like May. We are only interested in those which contain the highlighted anniversaries, as the one shown in the introduction. One way to expose these, is to filter the paragraphs down to ones that have the `'{Month} {day}: Event 1 (maybe some parenthesis); Event 2; Event 3, something, something'` structure

Skip to main content

```
<a href="/wiki/May_1" title="May 1">May 1</a>
```

If you have taken a good look at the html page yourself, you have probably convinced yourself already that all of the relevant paragraphs have an attribute like this one, and that the page url is of the format `/wiki/{Month}_{day}`. You should exploit these details.

Remember that regex can be a useful tool here!

Example: if you pass the html text for the May

```
ann_list = extract_anniversaries(html, month="May")
```

your anniversaries list `ann_list` may look like:

```
['May 1: Beltane and Samhain in the Northern and Southern Hemispheres, respectively
'May 2',
'May 3: World Press Freedom Day; Constitution Memorial Day in Japan (1947); Constit
'May 4: Youth Day in China; Literary Day in Taiwan; National Day of Prayer in the U
"May 5: Children's Day in Japan; Cinco de Mayo in Mexico and the United States (186
...
]
```

You can test your implementation by running:

```
pytest -v -m task31 tests
```

Given a list of anniversaries, create a dataframe

Now comes the final preparations, before you can explore all the anniversaries in a nicely formatted table.

> **Task 3.2 (3 points)** Inside `find_anniversaries.py`, complete the function `anniversary_list_to_df`, which should take a list of strings returned by `extract_anniversaries`, and split these into date and event parts, and return a dataframe with “Date” and “Event” columns.

Specifications

Skip to main content

- If there are multiple events highlighted on a specific date, each one of them must be stored as a separate entry in the dataframe, under the given date.
- There are some paragraphs that contain a date only, and no highlighted events, with or without a semicolon like:

  May 2

  - 1194 – King Richard I of England gave the city of **Portsmouth** *(Old Portsmouth pictured)* its first royal charter.

  June 16:

  - 1407 – Ming forces **conquered Đại Ngu** in modern-day northern Vietnam, capturing Hồ dynasty emperor Hồ Quý Ly and bringing

  The dataframe should not contain dates without events, so make sure to skip all paragraphs that look like this.

> **Hint**
>
> You may split the string in two parts separated with the colon (`:`) by using `str.partition()` method, and use regex to split the event part intro separate events. Want to practice even more regex? Use only regex to do all the splitting.

Note that

- Some events may have a parenthesis containing a colon, like for example `December solstice (21:48 UTC, 2022)` event on December 21st.
- There are also events where the semicolon appears as a part in the event description, within a pair of parentheses like this event on December 25th: `Christmas (Western Christianity; Gregorian calendar)`. Make sure to handle this correctly. Regex may be an advantage here, as you may adapt your regex pattern to ignore semicolons that appear between parentheses.
- There are a few paragraphs where two distinct events are separated by a comma instead of a semicolon, like for example `Laba Festival in China (2022), Rizal Day in the Philippines (1896)` at December 30th. We will ignore this special case, and not deduct points if you store these as one event in the final table.

You can test your implementation by running:

```
pytest -v -m task32 tests
```

Make an anniversary table

It is time to put it all together and create those beautiful tables!

Skip to main content

> **Task 3.3 (5 points)** Inside `find_anniversaries.py`, complete the function named `anniversary_table`, which should take a url to the the `Wikipedia:Selected_anniversaries` namespace, a list of months to investigate, and a path to your working directory. The function should create a markdown table for each month in the list and save it in a separate directory named `tables_of_anniversaries` in your current working directory. The table names must follow the pattern `"anniversaries_{month}.md"`, where `month` **should be lowercase**.

Steps that you will need to cover within this task are the following:

- Extract the html from the full url for each month-specific page
- Call `extract_anniversaries` function to get the list of anniversaries
- Call `anniversary_list_to_df` function to get the dataframe
- Convert the dataframe to markdown and save the output table to `anniversaries_{lowercase_month}.md`

For instance, the final markdown table for May page, will have the name `_anniversaries_may.md` and should look something like this:

```
| Date  | Event
| :---- | :-------------------------------------------------------------------
| May 1 | Beltane and Samhain in the Northern and Southern Hemispheres, respectivel
| May 1 | Maharashtra Day in Maharashtra, India (1960)
| May 1 | Loyalty Day in the United States
| May 3 | World Press Freedom Day
| May 3 | Constitution Memorial Day in Japan (1947)
| May 3 | Constitution Day in Poland (1791)
| May 4 | Youth Day in China
```

... with lots more rows that are not shown here.

Explore and save your results

Now you have laid an important groundwork and you are ready to explore all the months and their anniversaries!

Try to run with your birthday month, do you find any interesting events on your birthday date? Are there any funny, peculiar, or interesting events that you did not know about?

Skip to main content

months in the Wikipedia:Selected_anniversaries namespace, as a global variable named `months_in_namespace` at the beginning of the script. To pass this task, you must run `anniversary_table` with this list as a parameter, and create a table for each month in the namespace.

Files to deliver in this task: Create a directory `assignment4/tables_of_anniversaries` where you store all the files created in this task.

Files required in this task:

- `anniversaries_january.md`
- `anniversaries_february.md`
- `anniversaries_march.md`
- `anniversaries_april.md`
- `anniversaries_may.md`
- `anniversaries_june.md`
- `anniversaries_july.md`
- `anniversaries_august.md`
- `anniversaries_september.md`
- `anniversaries_october.md`
- `anniversaries_november.md`
- `anniversaries_december.md`

You can test your implementation by running:

```
pytest -v -m task33 tests
```

This test calls on `anniversary_table` in a temporary directory, and checks if the result contains the correctly named directory and all the necessary files with correct names in it. Make sure that this test passes before submitting this task.

Preparing for summer Olympic Games in Paris 2024

Excited as you are for the Summer Olympic Games coming up in Paris the summer of 2024, you really want to dive into the current state of the art. You are aware that Norway is

Skip to main content

you a bit nervous. What about our Scandinavian siblings, Denmark and Sweden? Can we expect them to be strong competitors? One way of finding out is to look back at the history.

In this task you are going to complete the script `fetch_olympic_statistics.py` which visits the All-time Olympic Games medal table article on Wikipedia, collects and plots some Olympic games statistics for the three Scandinavian countries **Norway, Denmark, and Sweden**. From this website, you are going to extract urls to the Scandinavian countries, which lead to an article like this: Norway at the Olympics, where you are going to extract statistics from the *Medals by summer sport* table.

This task can be broken down into four subtasks:

1. Given the All-time Olympic Games medal table article, get the urls and the total number of gold medals in summer and winter games for Norway, Denmark and Sweden.
2. Given the Norway at the Olympics article, and similarly for Denmark and Sweden, get the number of gold, silver and bronze medals for a given type of summer sport.
3. Given the acquired data, do analysis on what country is the best in a given summer sport
4. Plot the medal statistics as bar charts and create an .md table of the best country in each sport ranked by gold medals.

As in the previous task, we will use Beautiful Soup for parsing.

Before proceeding with the code, we strongly recommend you to inspect the abovementioned wikipedia articles and see how these tables are built up. Also, remember that you can search inside the code that you inspect within a webpage.

Get the urls and gold medal stats for Scandinavian countries, given a wiki url

> **Task 4.1 (3 points)** Inside `fetch_olympic_statistics.py`, complete the function named `get_scandi_stats`, which should take a url to the All-time Olympic Games medal table and return two dictionaries:
>
> - one with country names as keys and urls to the country specific performance pages as values
> - and the other with country names as keys, and a dictionary of summer and winter gold medals as values

Skip to main content

This function should be able to filter the search for the countries in a given list, and we have defined the `scandinavian_countries` list as a global variable in this script for this purpose.

`get_scandi_stats` should:

1. Locate and parse the *List of NOCs with medals* table
2. Filter the rows to the countries in the selected list
3. Return a dictionary: `country_dict` with URLs and gold medal counts by summer and winter

Output should look like this

```
{
  "Denmark": {
    "url": "https://en.wikipedia.org/wiki/Denmark_at_the_Olympics",
    "medals": {
      "Summer": 48,
      "Winter": 0,
    },
  },
  "Norway": {
      "url": "...",
```

You can test your implementation by running:

```
pytest -v -m task41 tests
```

Get the medal stats for the country, given a wiki url and a sport

Great work! Now, we have a url to the country specific performance page for Norway, Denmark, and Sweden, and we are ready to dig deeper in the statistics.

The tables we are interested in look like:

Skip to main content

Medals by summer sport [edit]

Sport	Gold	Silver	Bronze	Total
Sailing	17	11	4	32
Shooting	13	8	11	32
Athletics	8	7	8	23
Canoeing	6	4	4	14
Wrestling	4	2	2	8
Rowing	3	7	8	18
Handball	2	2	3	7
Cycling	2	0	2	4
Boxing	1	2	2	5
Gymnastics	1	2	1	4
Football	1	0	2	3
Beach volleyball	1	0	0	1
Triathlon	1	0	0	1
Weightlifting	1	0	0	1
Taekwondo	0	2	0	2
Swimming	0	1	1	2
Equestrian	0	1	0	1
Fencing	0	1	0	1
Tennis	0	0	1	1
Totals (19 entries)	**61**	**50**	**49**	**160**

> **Task 4.2 (3 points)** Inside `fetch_olympic_statistics.py`, complete the function named `get_sport_stats`, which should take a url to a country specific performance page in the Olympics and a summer sport name, and return a dictionary of the gold, silver, and bronze medals acquired by the country in the given sport. A given country might not compete in the sport you request. In such case, the function must return `{"Gold": 0, "Silver": 0, "Bronze": 0}`

For example:

```
get_sport_stats("https://en.wikipedia.org/w/index.php?title=Norway_at_the_Olympics&
```

Skip to main content

should return:

```
{"Gold": 17, "Silver": 11, "Bronze": 4}
```

Note

The `oldid` parameter is a permanent link, which means a specific version of the article in time. You don't need to use these, but they are useful in tests, so your tests will still pass after the next Olympic games.

Warning

Be careful with capital letters when searching for the *Medals by summer sport* table. Using regex might be a good choice here.

You can test your implementation by running:

```
pytest -v -m task42 tests
```

Get the best country in a given sport, given medal statistics

We want to compare Norway, Denmark, and Sweden, and it would be interesting to know which one (or more) of them is the best in a given summer sport.

> **Task 4.3 (2 points)** Inside `fetch_olympic_statistics.py`, complete the function named `find_best_country_in_sport`, which takes the following:
>
> - results - a dictionary of country specific medal results in a given sport, having this form:
>
> ```
> {
>  "Norway" : {"Gold" : 2, "Silver" : 1, "Bronze" : 3},
>  "Sweden" : {"Gold" : 1, "Silver" : 2, "Bronze" : 3},
>  "Denmark" : {"Gold" : 1, "Silver" : 2, "Bronze" : 3},
> }
> ```

Skip to main content

> - `medal` - medal type to compare for. Valid parameters: `"Gold" | "Silver" | "Bronze"`
>
> ... and returns the name(s) of the best country in the given sport as a string, based on the highest number of the medal acquired, of the given type.

For example, given

```
results = {
 "Norway" : {"Gold" : 2, "Silver" : 1, "Bronze" : 3},
 "Sweden" : {"Gold" : 1, "Silver" : 2, "Bronze" : 3},
 "Denmark" : {"Gold" : 1, "Silver" : 2, "Bronze" : 3},
}
```

it should return `"Norway"` for "Gold", `"Sweden/Denmark"` for "Silver" (order not important), and `"None"` (string, not NoneType) for "Bronze".

You can test your implementation by running:

```
pytest -v -m task43 tests
```

Comparing and plotting Olympic statistics

At this point, you have the tools to acquire statistics for any selected set of countries competing in the Olympics. That is a lot of data to deal with!

To inform yourself of the current state of the games, and to lay the ground for your future predictions with regard to Norway, Denmark, and Sweden, you will now visualize the statistics in different ways.

We have provided you a list of summer sports as a global variable `summer_sports` in the script. These are the summer Olympic sports you must include. However, you can *add* to the list list and run the script with other sports as well, if you are interested.

The following is a list of filenames and data the corresponding plots must exhibit:

- `total_medal_ranking.png` - a bar chart displaying the *total* number of gold medals in summer and winter games for the three Scandinavian countries.
- `{sport}_medal_ranking.png` - a bar chart displaying the total number of gold, silver and bronze medals in a given sport for the three Scandinavian countries. You must produce

Skip to main content

one such plot for each of the sports in the `summer_sports` list. Thus, `{sport}` must take the value of one summer Olympic sport, for example `Sailing`.

- `best_of_sport_by_Gold.md` - a markdown table with columns `Sport` and `Best country` displaying the best country in each of the sports in the `summer_sports` list.

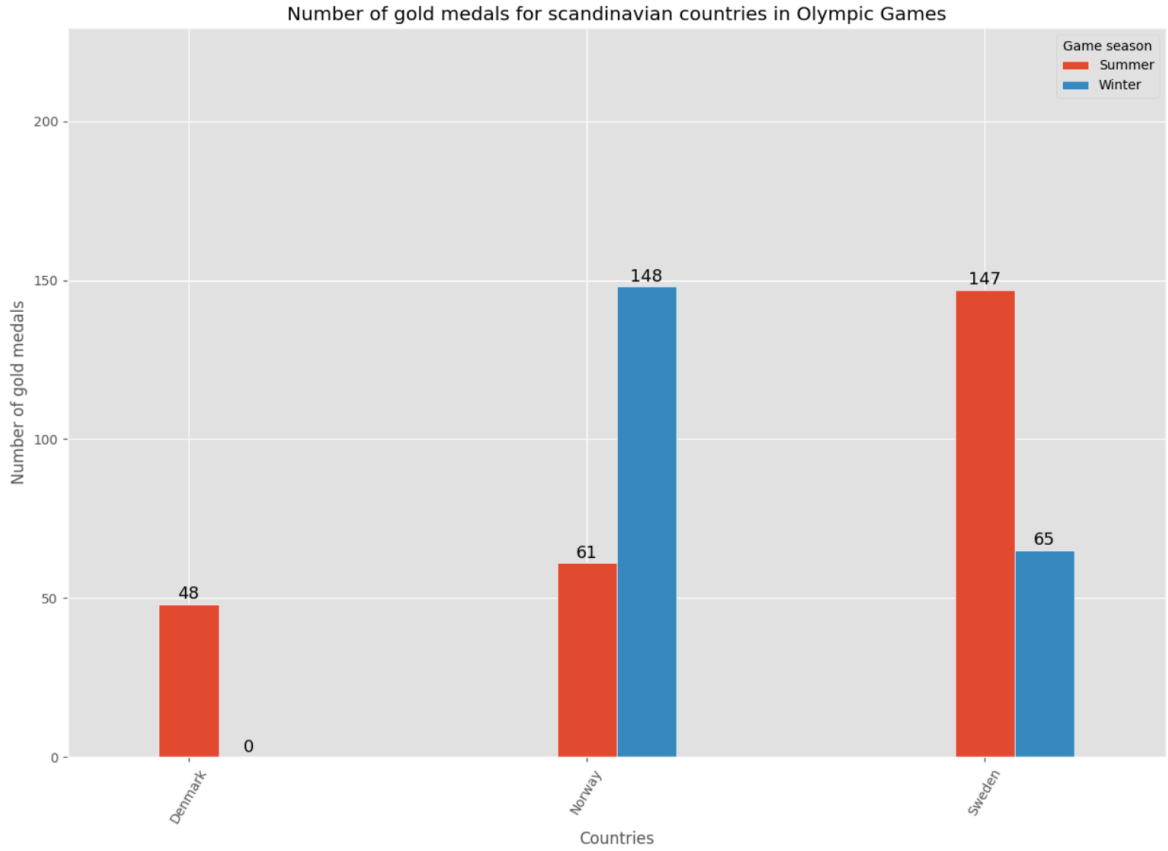
We will test your code in a temporary directory, where these plots will be saved, so make sure that the names of the plots are exactly the same as requested above. A complete list of files that must be submitted is provided at the end of this task.

> **Task 4.4 (5 points)** Inside `fetch_olympic_statistics.py`, tie task 4 together in the function named `report_scandi_stats` that takes a url to the All-time Olympic Games medal table a list of sports and the path to your current working directory. The function should:
>
> 1. Call `get_scandi_stats` to get the urls for the countries of interest and their total number of medals
> 2. Create and save the `total_medal_ranking.png` statistics
> 3. Iterate through each sport and make a call to `get_sport_stats`
> 4. Create and save the `{sport}_medal_ranking.png` statistics
> 5. Make a call to `find_best_country_in_sport` for each sport
> 6. Create and save the .md table of best country in each sport, named `best_of_sport_by_Gold.md`

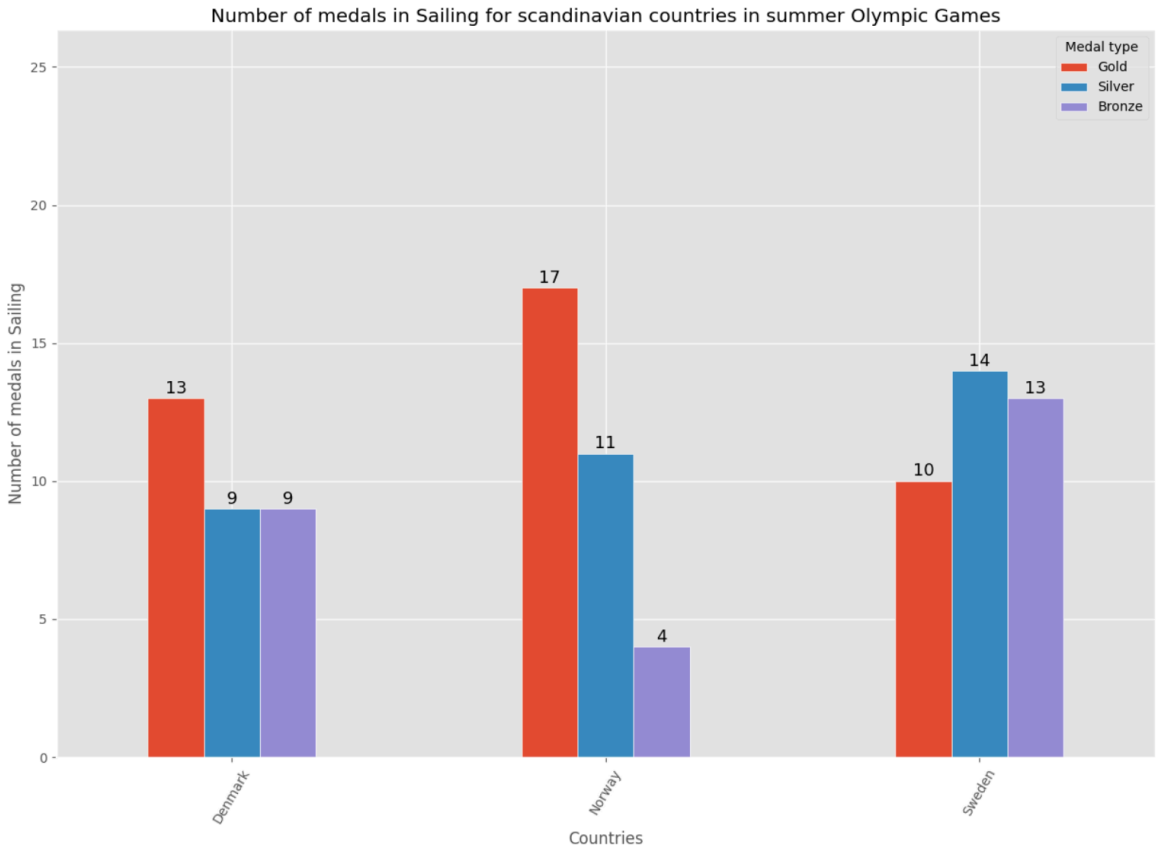
There are lots of ways to do the plotting. We have provided a sample plotting function in `example-plot.py`, which uses matplotlib. You can use this for a reference.

Skip to main content

We also provide you with an example of `total_medal_ranking.png`:



And of `Sailing_medal_ranking.png`:



The table `best_of_sport_by_Gold.md` might look like this:

Skip to main content

```
Best Scandinavian country in Summer Olympic sports, based on most number of Gold me
| Sport     | Best country   |
|:----------|:---------------|
| Sailing   | Norway         |
| Athletics | Sweden         |
| Handball  | Denmark        |
```

You can test your implementation by running:

```
pytest -v -m task44 tests
```

Files to deliver in this subtask: Create a directory `assignment4/olympic_games_results` where you store all the files created in this task.

Files required in this task:

- `Archery_medal_ranking.png`
- `Athletics_medal_ranking.png`
- `Cycling_medal_ranking.png`
- `Football_medal_ranking.png`
- `Handball_medal_ranking.png`
- `Sailing_medal_ranking.png`
- `total_medal_ranking.png`
- `best_of_sport_by_Gold.md`

Before you submit

If you want to check if you have all the necessary files in your `assignment4` directory before submission, type `pytest -vv tests/test_files.py` to run the test script we have provided, that does these checks for you and outputs a message if a file is missing.

Challenge - Wiki Race with URLs (5 bonus points)

You have probably heard of golf. You try to get the ball into the hole with the least amount of hits. Let's play some Wikipedia golf.

Skip to main content

Write a function using what you've done in the first tasks that finds the shortest path (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 function should work with any two wikipedia urls.

```
from wiki_race_challenge import find_path
path = find_path(start, finish)
assert path[0] == start
assert path[-1] == finish

print(f"Got from {start} to {finish} in {len(urls)-1} links")
```

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

Points are given for writing a script that finds the shortest path in a finite amount of time. You can test your implementation by finding 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.

Note: You are going to be on English wikipedia en.wikipedia.org.

Files to Deliver in this Subtask

- `wiki_race_challenge.py`

Turning it in

- Remember to push to `https://github.uio.no/IN3110/IN3110-yourname`. **Not your fork at `https://github.uio.no/yourname/IN3110-yourname`!**
- All files should be in the folder `assignment4/` (we will not grade files delivered elsewhere)
- As always, extensions can be requested with this form, and Devilry will only be used to return grades, it will not reflect extensions.
- You should have these files (exact names required):
  - assignment4/requesting_urls.py
  - assignment4/filter_urls.py

Skip to main content

    - assignment4/find_anniversaries.py
    - assignment4/fetch_olympic_statistics.py
    - assignment4/wiki_race_challenge.py (optional)
- And the directories containing files specified in task 3, 4 (exact names required):
    - assignment4/tables_of_anniversaries
    - assignment4/olympic_games_results
- All tests in the tests directory (`pytest -vv tests`) should pass without modification. Passing all of these tests is necessary to receive full credit.

Previous
Assignment 3: Python for the gram

Next
Assignment 5: Strømpris