Webscraping II

Peter Ganong and Maggie Shi

February 4, 2026

Table of contents I

Can I Scrape It?

Scraping Data from Tables

Web Crawlers

Recap and skills you'll acquire this lecture

Last class, we covered:

- Intro to HTML: tags, attributes, and content
- Using `BeautifulSoup` to parse HTML and extract text

Recap and skills you'll acquire this lecture

Last class, we covered:

- Intro to HTML: tags, attributes, and content
- Using `BeautifulSoup` to parse HTML and extract text

Today, we'll cover:

- Understand webscraping etiquette
- Understand how to extract data from *tables* on webpages
- Learn how to use scraping to dynamically 'crawl' the web

Can I Scrape It?

Can I Scrape It?: Roadmap

- Discuss cases where websites block scraping
- Discuss solutions/workarounds
- How to determine if a site is scrapable ahead of time

Websites Can Block Scraping: Examples

- Many modern websites have built in mechanisms to prevent scraping

```
url = "https://www.amazon.com/"
response = requests.get(url)
soup = BeautifulSoup(response.content, 'lxml')
soup.text[0:100]
```

```
'\n\n\n\n\nSorry! Something went wrong!\n\n\n\n
```

Websites Can Block Scraping: Examples

- Many modern websites have built in mechanisms to prevent scraping

```
url = "https://www.amazon.com/"
response = requests.get(url)
soup = BeautifulSoup(response.content, 'lxml')
soup.text[0:100]
```

```
'\n\n\n\n\nSorry! Something went wrong!\n\n\n\n
```

Websites Can Reqiure You to Identify Yourself

```
url = "https://en.wikipedia.org/wiki/Web_scraping"

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

print(soup.text[:200])
```

```
Please set a user-agent and respect our robot policy https://w.wiki/4wJS. S
```

Websites Can Require You to Identify Yourself

- A **User-Agent** header tells the server who is making the request and how to handle it
  - Some sites, like Wikipedia, block requests with missing or generic User-Agents
- Components:
  - **Application name** — identifies your script or project
  - **Version number** — helps distinguish updates to your tool
  - **Contact information** — email or URL so site admins can reach you

Websites Can Require You to Identify Yourself

You define headers in Python using a dictionary passed to `requests.get()`

```
url = "https://en.wikipedia.org/wiki/Web_scraping"

myheader = {"User-Agent": "DAP30538CourseBot/1.0
↪  (m.shi@uchicago.edu)"}

response = requests.get(url, headers=myheader)
soup = BeautifulSoup(response.text, "lxml")

print(soup.text[:200])
```

```
Web scraping - Wikipedia
```

Websites Can Block Crawlers: Rate-Limiting

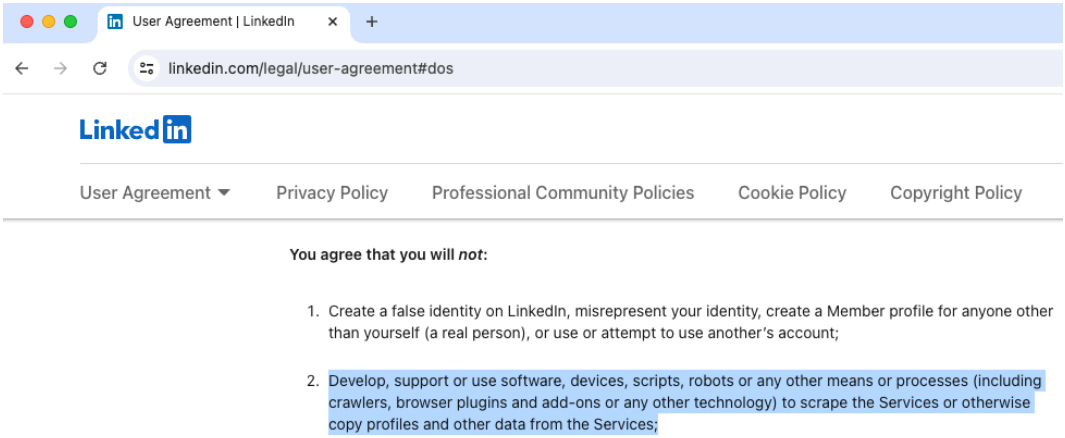
- Websites can also 'rate-limit': track how often a bot accesses the site and block it if it access it too often
- In this case, your request will return a `429 Too Many Requests` error
- Can avoid this by adding a delay with `time.sleep()` in between iterations

```
import time

for url in urls:
    response = requests.get(url)
    time.sleep(2)  # Add a 2-second delay
```

Can I Scrape It?

1. Check Terms of Service



Can I Scrape It?

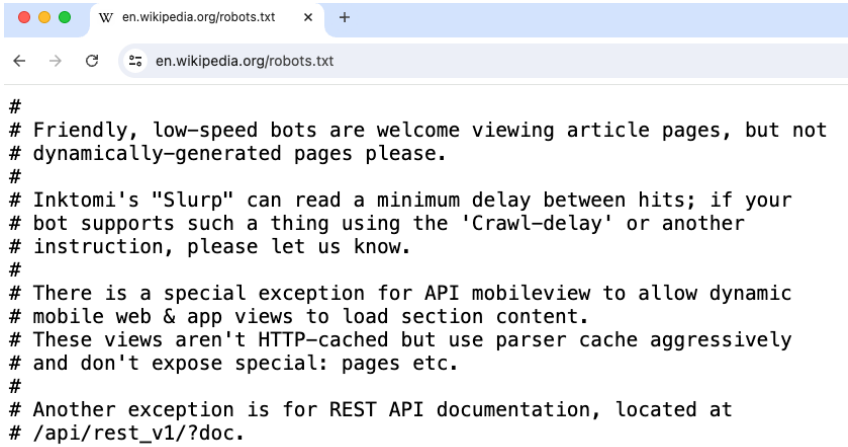
2. Check `robots.txt`

```
# Notice: The use of robots or other automated means to access LinkedIn without
# the express permission of LinkedIn is strictly prohibited.
# See https://www.linkedin.com/legal/user-agreement.
# LinkedIn may, in its discretion, permit certain automated access to certain LinkedIn pages,
# for the limited purpose of including content in approved publicly available search engines.
# If you would like to apply for permission to crawl LinkedIn, please email whitelist-crawl@linkedin.com.
# Any and all permitted crawling of LinkedIn is subject to LinkedIn's Crawling Terms and Conditions.
# See http://www.linkedin.com/legal/crawling-terms.

User-agent: LinkedInBot
Allow: /
```

Can I Scrape It?

2. Check `robots.txt`



Can I Scrape It?: Summary

- Some sites block all scraping
- Others require you to declare your identity or will rate-limit you
- Check Terms of Service or `robots.txt` to see a website's policy on scraping

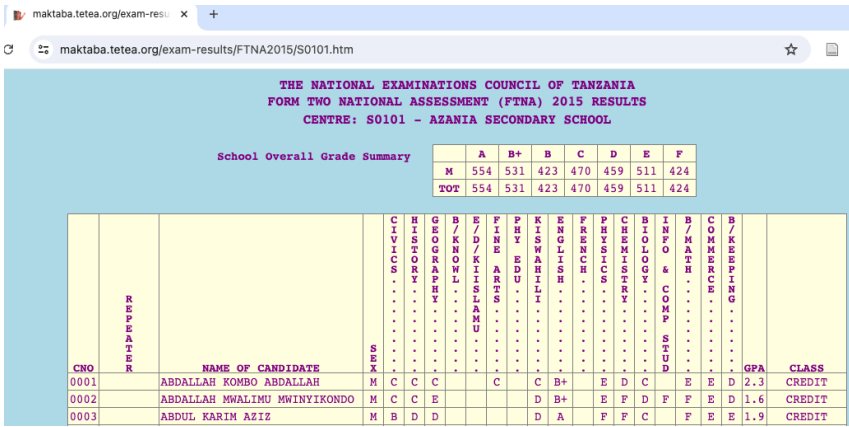
Scraping Data from Tables

Extracting Data from Tables: Intro and Roadmap

- One important use case for web scraping is to **automatically extract and save data from tables**
- When should I scrape a table?
- Walk through example: national exam results from Tanzania

Example: A Scrapable Table

- Example: Tanzania National Examination outcomes (link)



maktaba.tetea.org/exam-resu

maktaba.tetea.org/exam-results/FTNA2015/S0101.htm

THE NATIONAL EXAMINATIONS COUNCIL OF TANZANIA
FORM TWO NATIONAL ASSESSMENT (FTNA) 2015 RESULTS
CENTRE: S0101 - AZANIA SECONDARY SCHOOL

School Overall Grade Summary

	A	B+	B	C	D	E	F
M	554	531	423	470	459	511	424
TOT	554	531	423	470	459	511	424

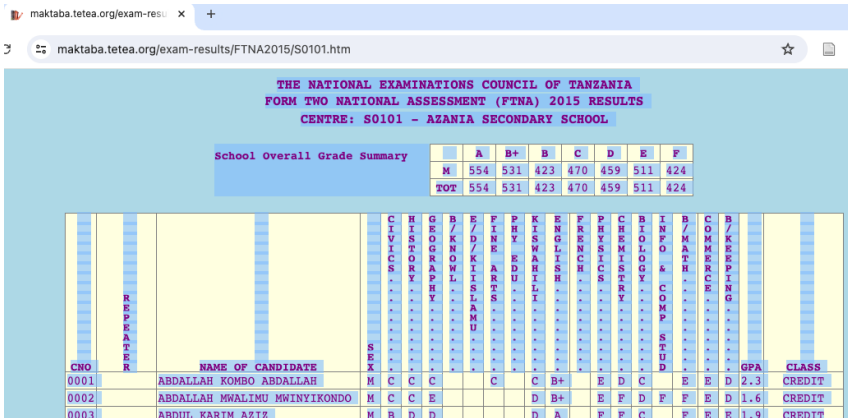
CNO	REPEATER	NAME OF CANDIDATE	SEX	CIVICS	HISTORY	GEOGRAPHY	B/KNOWL	ED/KISWAHILI	FINE ARTS	PHY EDU	KISWAHILI	ENGLISH	FRENCH	PHYSICS	CHEMISTRY	BIOLOGY	INFO & COMP STUD	B/MATH	COMMERCE	B/KEEPING	GPA	CLASS
0001		ABDALLAH KOMBO ABDALLAH	M	C	C	C			C		C	B+		E	D	C		E	E	D	2.3	CREDIT
0002		ABDALLAH MWALIMU MWINYIKONDO	M	C	C	E					D	B+		E	F	D	F	F	E	D	1.6	CREDIT
0003		ABDUL KARIM AZIZ	M	B	D	D					D	A		F	F	C		F	E	E	1.9	CREDIT

When to Scrape?

- If your goal was just to scrape this particular table one time, copy-and-pasting is fine.
- You should webscrape if:
  - Your tables are spread across multiple pages: example with Tanzania (link)
  - Data tables are using "lazy-loading", so you have to scroll over and over again to "Show More": example with ESPN (link)
  - In these cases, you actually probably want to web-*crawl* (discussed next)
- Webscraping should be a tool of last resort!
  - If there is an API, use it
  - If there is a portal to download data, use it

Example: A Scrapable Table

- Example: Tanzania National Examination outcomes (link)



maktaba.tetea.org/exam-resu

maktaba.tetea.org/exam-results/FTNA2015/S0101.htm

THE NATIONAL EXAMINATIONS COUNCIL OF TANZANIA
FORM TWO NATIONAL ASSESSMENT (FTNA) 2015 RESULTS
CENTRE: S0101 - AZANIA SECONDARY SCHOOL

School Overall Grade Summary

	A	B+	B	C	D	E	F
M	554	531	423	470	459	511	424
TOT	554	531	423	470	459	511	424

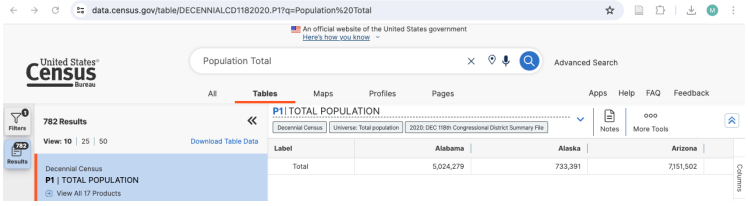
CNO	REPEATER	NAME OF CANDIDATE	SEX	CIVICS	HISTORY	GEOGRAPHY	BK/KISWAHILI	ED/KISLAMU	FINE ARTS	EDU.	PHY	KISWAHILI	ENGLISH	FRENCH	PHYSICS	CHEMISTRY	BIOLOGY	INFO & COMP STUD	BK/MATH	COMMERCE	BK/KEEPING	GPA	CLASS
0001		ABDALLAH KOMBO ABDALLAH	M	C	C	C			C		C	B+		E	D	C		E	E	D	2.3	CREDIT
0002		ABDALLAH MWALIMU MWINYIKONDO	M	C	C	E					D	B+		E	F	D	F	F	E	D	1.6	CREDIT
0003		ABDUL KARIM AZIZ	M	B	D	D					D	A		F	F	C		F	E	E	1.9	CREDIT

- This table is a good candidate for scraping: it is static and the data appear in the HTML code

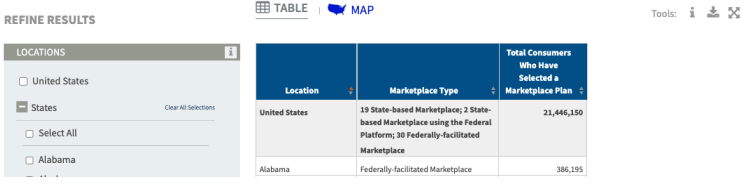
Counterexample: Data Viewers

We would not recommend scraping dynamic tables/"data viewers"

Census "Explore Census Data" (link)



Kaiser Family Foundation "State Health Facts" (link)

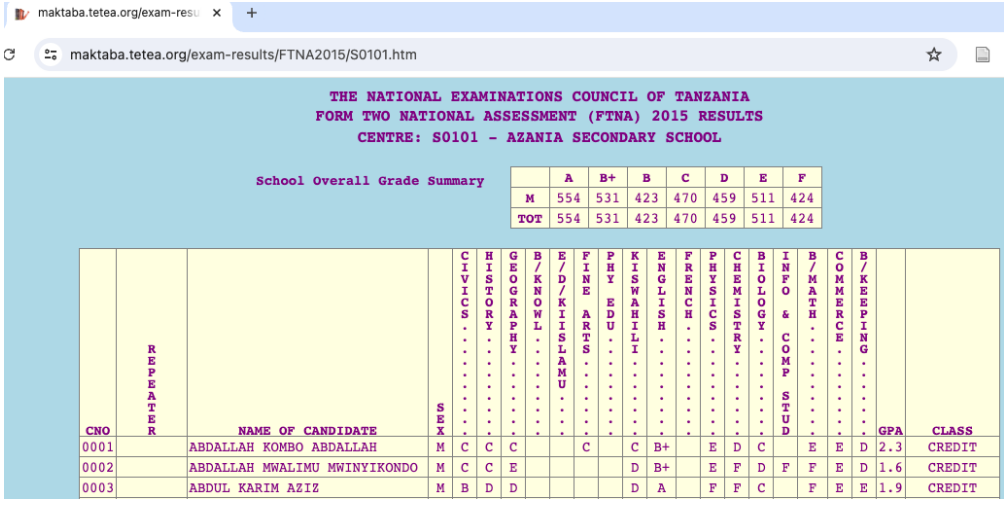


Example: Data Viewers

- In contrast, would not recommend scraping dynamic tables/"data viewers"
- These often dynamically filter the data, rather than displaying it all in HTML
- And they typically offer other options to access the data!
- If you have to scrape these more advance sites, you could use the `selenium` package, which allows you to control a web browser through Python

Example: Scraping a Table

- Let's try to scrape the table on this webpage



THE NATIONAL EXAMINATIONS COUNCIL OF TANZANIA
FORM TWO NATIONAL ASSESSMENT (FTNA) 2015 RESULTS
CENTRE: S0101 - AZANIA SECONDARY SCHOOL

School Overall Grade Summary

	A	B+	B	C	D	E	F
M	554	531	423	470	459	511	424
TOT	554	531	423	470	459	511	424

CNO	REPEATER	NAME OF CANDIDATE	SEX	CIVICS	HISTORY	GEOGRAPHY	B/KNOWL	E/D/KISLAMU	FINE ARTS	PHY EDU	KISWAHILI	ENGLISH	FRENCH	PHYSICS	CHEMISTRY	BIOLOGY	INFO & COMP STUD	B/MATH	COMMERCE	B/KEEPING	GPA	CLASS
0001		ABDALLAH KOMBO ABDALLAH	M	C	C	C			C		C	B+		E	D	C		E	E	D	2.3	CREDIT
0002		ABDALLAH MWALIMU MWINYIKONDO	M	C	C	E					D	B+		E	F	D	F	F	E	D	1.6	CREDIT
0003		ABDUL KARIM AZIZ	M	B	D	D					D	A		F	F	C		F	E	E	1.9	CREDIT

Side Note on AI

- Note: as of January 2026, ChatGPT could not write code that runs "out-of-the-box" to scrape this site (Link to response)
- It misses the fact that each column is a different class

The DataFrame will look like:

cno	name	repeater	sex	grades_raw	gpa	class
0001	ABDALLAH KOMBO ABDALLAH	False	M	C C C B C C C C B B	2.3	CREDIT
0002	...	...	...	...	...	...

  - You could, however, use its response as a jumping off point for developing and refining your scraper

Example: Scraping a Table

Recall that the steps of building a webscraper are:

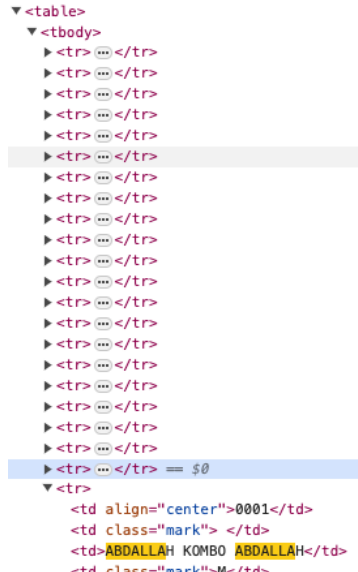
1. *Manual*: inspect website's HTML to see how the info we want to extract is structured and how a computer/scraper sees the webpage
2. *Code*: download and save HTML associated with a website
3. *Code*: extract information you want to scrape from HTML

Do-pair-share: Step 1. Inspect website and HTML

1. Go to the Tanzania National Examination outcomes (link)
2. Familiarize with the web page and table – **what is the unit of observation**?
3. From looking at the table, **name a detail** about the structure and layout of the table that you anticipate could make scraping difficult
4. Right click over one of the observations and "Inspect" it: **what tag(s) does the data appear to be stored in**?

Step 1. Inspect website and HTML

- Given the issue with the header, we're going to start with the first row with a name
- Search for the name in the first row: 'ABDALLAH'
- Appears to be in a nested tag: `<table>` → `<tbody>` → `<tr>` → `<td>`
  - Note: `<tbody>` is not in the actual raw HTML file, but is instead added by the browser to automatically fix incorrect HTML code
  - `BeautifulSoup`, however, reads in the actual (uncorrected) source code



Step 2. Download and save HTML

- Start with the usual: making a request to website and parsing the response with `BeautifulSoup`

```
url = 'https://maktaba.tetea.org/exam-results/FTNA2015/S0101.htm'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'lxml')
```

Step 3. Extract + refine

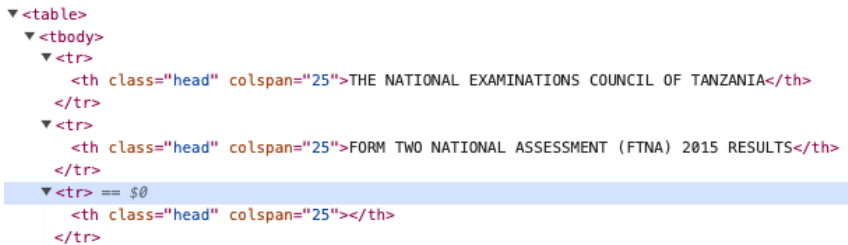
- Let's use `.find` to find the *first* instance of the `table` tag
- Then apply `.find_all` to see what's inside that table

```
table = soup.find('table')
rows = table.find_all('tr') #tr stands for table row
rows[0:5]
```

```
[<tr><th class="head" colspan="25">THE NATIONAL EXAMINATIONS COUNCIL OF TA
 <tr><th class="head" colspan="25">FORM TWO NATIONAL ASSESSMENT (FTNA) 201
 <tr><th class="head" colspan="25"></th></tr>,
 <tr><th class="head" colspan="25">CENTRE: S0101 - AZANIA SECONDARY SCHOOL
 <tr><th colspan="25" style="border-left-color:#ADD8E6;border-right-color:
 <table class="top_sum">
 <tr style="font-weight:bold;"><td> </td><td>A</td><td>B+</td><td>B</td><t
```

Step 3. Extract + refine

- The output from looking for the `rows` tag aligns with the HTML code



- But we don't want to scrape the first lines or the header
- At this point, we should look through the items in `rows`

Step 3. Extract + refine

- After a bit of searching through the `rows` object, we will find that the first row starts in index 24

```
rows[24]
```

```
<tr><td align="center">0001</td><td class="mark"> </td><td>ABDALLAH KOMBO
```

- We're almost at the data!
- But need to drill down one more level to each cell, which are denoted by `td`

Step 3. Extract + refine

- Apply `.find_all()` again to look for `td`

```
cells = rows[24].find_all('td')
cells[0:3]
```

```
[<td align="center">0001</td>,
 <td class="mark"> </td>,
 <td>ABDALLAH KOMBO ABDALLAH</td>]
```

Step 3. Extract + refine

- Apply `.find_all()` again to look for `td`

```
cells = rows[24].find_all('td')
cells[0:3]
```

```
[<td align="center">0001</td>,
 <td class="mark"> </td>,
 <td>ABDALLAH KOMBO ABDALLAH</td>]
```

```
cells[0].text
```

```
'0001'
```

```
cells[2].text
```

```
'ABDALLAH KOMBO ABDALLAH'
```

Step 3. Extract + refine

- To extract information, we need *two* nested for loops. Our final scraping code:

```
data_rows = []

for row in table.find_all('tr')[24:]:                #1
    cells = row.find_all('td')                       #2
    data_rows.append([val.text for val in cells])    #3
```

Step 3. Extract + refine

- To extract information, we need *two* nested for loops. Our final scraping code:

```
data_rows = []

for row in table.find_all('tr')[24:]:                #1
    cells = row.find_all('td')                       #2
    data_rows.append([val.text for val in cells])    #3
```

1. Loops through all `tr` tags in `table` (after the 24th row)
2. Within each `tr` tag in `rows`, finds all the `td` tags
3. Loops through `cell` and appends text to `data_rows`

Post-scraping cleanup: organize and save extracted information

- `data_rows` is a list, so we need to convert it into a dataframe

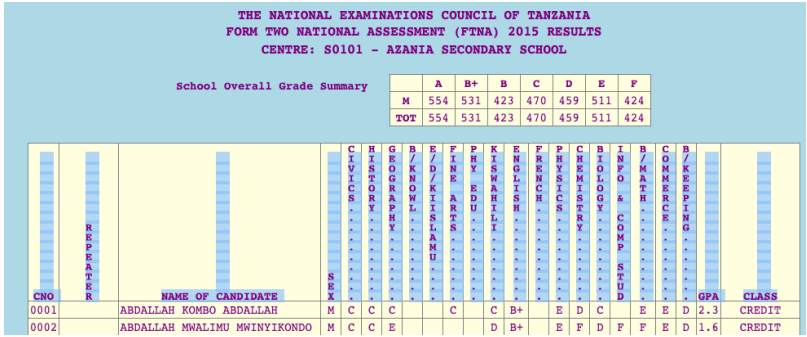
```
data_rows = pd.DataFrame(data_rows)
print(data_rows.head())
```

```
     0  1                                2  3  4  5   6  7  8  9   ...  13 14
0  0001          ABDALLAH KOMBO ABDALLAH  M  C  C   C        C  ...      E
1  0002     ABDALLAH MWALIMU MWINYIKONDO  M  C  C   E           ...      E
2  0003                ABDUL KARIM AZIZ  M  B  D   D           ...      F
3  0004               ABDUL SEIF MBONDE  M  A  A   A           ...  B+  A
4  0005       ABDULATIFU HARUNI MAGIMILA  M  C  C  B+        C  ...      D

   16 17 18  19 20   21           22
0   C     E   E  D  2.3       CREDIT
1   D  F  F   E  D  1.6       CREDIT
2   C     F   E  E  1.9       CREDIT
3   A     A  B+  A  5.0  DISTINCTION
```

Post-scraping cleanup: organize and save extracted information

- We forgot the header!
- Can/should we scrape the header from the table?



THE NATIONAL EXAMINATIONS COUNCIL OF TANZANIA
FORM TWO NATIONAL ASSESSMENT (FTNA) 2015 RESULTS
CENTRE: S0101 - AZANIA SECONDARY SCHOOL

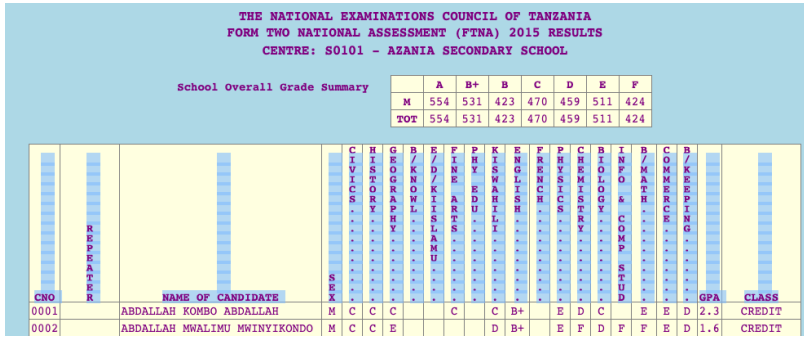
School Overall Grade Summary

	A	B+	B	C	D	E	F
M	554	531	423	470	459	511	424
TOT	554	531	423	470	459	511	424

CNO	REPEATER	NAME OF CANDIDATE	SEX	CIVICS	HISTORY	GEOGRAPHY	B/KNOWL	ED/KISWAHILI	FINE ARTS	PHY EDU	KISWAHILI	ENGLISH	FRENCH	PHYSICS	CHEMISTRY	BIOLOGY	INFO & COMP STUD	B/MATH	COMMERCE	B/KEEPING	GPA	CLASS
0001		ABDALLAH KOMBO ABDALLAH	M	C	C	C			C		C	B+		E	D	C		E	E	D	2.3	CREDIT
0002		ABDALLAH MWALIMU MWINYIKONDO	M	C	C	E					D	B+		E	F	D	F	F	E	D	1.6	CREDIT

Post-scraping cleanup: organize and save extracted information

- We forgot the header!
- Can/should we scrape the header from the table?



THE NATIONAL EXAMINATIONS COUNCIL OF TANZANIA
FORM TWO NATIONAL ASSESSMENT (FTNA) 2015 RESULTS
CENTRE: S0101 - AZANIA SECONDARY SCHOOL

School Overall Grade Summary

	A	B+	B	C	D	E	F
M	554	531	423	470	459	511	424
TOT	554	531	423	470	459	511	424

CNO	REPEATER	NAME OF CANDIDATE	SEX	CIVICS	HISTORY	GEOGRAPHY	KNOWL	ED/KISLAMU	FINE ARTS	PHY EDU	KISWAHILI	ENGLISH	FRENCH	PHYSICS	CHEMISTRY	BIOLOGY	INFO & COMP STUD	B/MATH	COMMERCE	B/KEEPING	GPA	CLASS
0001		ABDALLAH KOMBO ABDALLAH	M	C	C	C			C		C	B+		E	D	C		E	E	D	2.3	CREDIT
0002		ABDALLAH MWALIMU MWINYIKONDO	M	C	C	E					D	B+		E	F	D	F	F	E	D	1.6	CREDIT

- Given that the header is spread out across several rows and is sometimes vertical, scraping it would be a hassle

Post-scraping cleanup: fix header

- Instead of scraping it, we'll just hard-code it: define the header manually

```
my_cols = ['CNO', 'Repeater', 'Name', 'Sex', 'Civics', 'History',
 ↪  'Geography', 'Knowledge', 'EDK', 'Fine_Arts', 'Phys_Ed',
 'Kiswahili', 'English', 'French', 'Physics', 'Chemistry', 'Biology',
  ↪  'Info_Computer', 'Math', 'Commerice', 'Keeping', 'GPA', 'Class']

data_rows.columns = my_cols
```

- There's a tradeoff with webscraping: if you are doing something *just once*, probably not worth it to do it via scraping

Post-scraping cleanup: organize and save extracted information

```
print(data_rows.head())
```

```
    CNO Repeater                          Name Sex Civics History Geography  \
0  0001          ABDALLAH KOMBO ABDALLAH   M      C       C         C
1  0002     ABDALLAH MWALIMU MWINYIKONDO   M      C       C         E
2  0003               ABDUL KARIM AZIZ   M      B       D         D
3  0004              ABDUL SEIF MBONDE   M      A       A         A
4  0005       ABDULATIFU HARUNI MAGIMILA   M      C       C        B+

  Knowledge EDK Fine_Arts  ... French Physics Chemistry Biology Info_Computer  \
0                       C  ...              E         D       C
1                          ...              E         F       D             F
2                          ...              F         F       C
3                          ...     B+       A         A       A
4                       C  ...              D         B      B+

  Math Commerice Keeping  GPA        Class
0    E         E       D  2.3       CREDIT
1    F         F       D  1.6       CREDIT
```

Back to step 1: check original table again

CNO	REPEATER	NAME OF CANDIDATE	SEX	CIVICS	HISTORY	GEOGRAPHY	B/KNOWL	ED/KIISLAMU	FINE ARTS	PHY EDU	KISWAHILI	ENGLISH	FRENCH	PHYSICS	CHEMISTRY	BIOLOGY	INFO & COMP STUD	B/MATH	COMMERCE	B/KEEPING	GPA	CLASS
0001		ABDALLAH KOMBO ABDALLAH	M	C	C	C			C		C	B+		E	D	C		E	E	D	2.3	CREDIT
0002		ABDALLAH MWALIMU MWINYIKONDO	M	C	C	E					D	B+		E	F	D	F	F	E	D	1.6	CREDIT
0003		ABDUL KARIM AZIZ	M	B	D	D					D	A		F	F	C		F	E	E	1.9	CREDIT
0004		ABDUL SEIF MBONDE	M	A	A	A					B+	A	B+	A	A	A		A	B+	A	5.0	DISTINCTION
0005		ABDULATIFU HARUNI MAGIMILA	M	C	C	B+			C		C	A		D	B	B+		C	D	C	3.1	MERIT
0006		ABDULAZIZ HEBERT NCHIRA	M	B	C	B		C			B	B+		D	E	B		D	D	E	2.9	MERIT
0007		ABDULLATIF KHAMIS SAID	M	B+	D	B					C	B+	E	E	D	C		F	E	D	2.4	CREDIT
0008		ABDULATIFU FADHIL TAMBA	M	B+	C	A		E			C	A		B+	B	B		D	B	B	3.9	DISTINCTION
0009		ABDULMUTWALIB HAMIS TEMBO	M	A	B+	F					B	A	E	B	C	A		B	B+	B+	4.3	DISTINCTION
0010		ABUBAKARI EBRAHIM ABDALLAH	M	D	E	D					E	A	F	E	E	C		F	E	E	1.3	PASS
0011		ABUBAKARI ISSA KADEGE	M	B	D	B					C	B+		D	B	B	E	D	D	B+	3.1	MERIT
0012	REPEATER	ABUU ALLY JUMA	M	Absent																		ABS

Looking back at the original table, it looks like it should say `'Absent'` for each grade and `'ABS'` for GPA and class

Post-scraping cleanup: check subset of students who were Absent

Instead, we're seeing `'Absent'` under `'Civics'` and `None` for other grades

```
absent_subset = data_rows[data_rows['Civics'] == "Absent"]
print(absent_subset.head())
```

```
      CNO  Repeater                  Name Sex  Civics History Geography  \
11   0012  REPEATER          ABUU ALLY JUMA   M  Absent     ABS      None
71   0072  REPEATER        ELTON A MARANDU   M  Absent     ABS      None
86   0087                FAHAD HAMIS ALLY   M  Absent     ABS      None
171  0172            KELVIN GERALD SHILIMA   M  Absent     ABS      None
192  0193             MAULIDI HAJI SWALEHE   M  Absent     ABS      None

    Knowledge   EDK Fine_Arts  ... French Physics Chemistry Biology  \
11       None  None      None  ...   None    None      None    None
71       None  None      None  ...   None    None      None    None
86       None  None      None  ...   None    None      None    None
171      None  None      None  ...   None    None      None    None
192      None  None      None  ...   None    None      None    None
```

Post-scraping cleanup: replace grades of absent students

The rest of the data cleaning can be done by cleaning dataframes in pandas

```
data_rows.loc[data_rows['Civics'] == "Absent",
 ↪  data_rows.columns[4:-2]] = "Absent"

data_rows.loc[data_rows['Civics'] == "Absent",
 ↪  data_rows.columns[-2:]] = "ABS"
```

Post-scraping cleanup: check subset of students who were absent

```
absent_subset = data_rows[data_rows['Civics'] == "Absent"]
print(absent_subset.head())
```

```
      CNO  Repeater                    Name Sex  Civics History Geography  \
11   0012  REPEATER          ABUU ALLY JUMA   M  Absent  Absent    Absent
71   0072  REPEATER         ELTON A MARANDU   M  Absent  Absent    Absent
86   0087                  FAHAD HAMIS ALLY   M  Absent  Absent    Absent
171  0172            KELVIN GERALD SHILIMA   M  Absent  Absent    Absent
192  0193             MAULIDI HAJI SWALEHE   M  Absent  Absent    Absent

    Knowledge     EDK Fine_Arts  ...  French Physics Chemistry Biology  \
11     Absent  Absent    Absent  ...  Absent  Absent    Absent  Absent
71     Absent  Absent    Absent  ...  Absent  Absent    Absent  Absent
86     Absent  Absent    Absent  ...  Absent  Absent    Absent  Absent
171    Absent  Absent    Absent  ...  Absent  Absent    Absent  Absent
192    Absent  Absent    Absent  ...  Absent  Absent    Absent  Absent
```

Extracting Data from Tables: Summary

- `BeautifulSoup` can scrape HTML tables (but not dynamic JavaScript ones)
- HTML tables will be in `table` tag
  - `tr`: a row
  - `td`: a cell
- Most of the work will be in:
  - Setting up `BeautifulSoup` `.find_all()` to scrape what you want from the table
  - Cleaning up the scraper's mistakes with `pandas`
- May have to do several loops back to step 1 (manual inspection) to account for edge cases

Web Crawlers

Roadmap

- One powerful way to harness web scraping tools is to use them to 'crawl' the web – that is, to scrape and then automatically navigate from page to page
- Discuss use cases for web crawling
- Walk through a toy example of developing a Wikipedia web crawler

Use cases for web crawling

- Web crawl when you want to visit multiple URLs
  - Don't have a list of the URLs beforehand
  - But you know that they're stored in a structured or repetitive way
- You should web crawl when you don't know the exact URLs you want to visit, but know that *they link to each other*

Example: scraping from a list of press releases

WHO Africa

News Releases

Scaling up response to curb growing mpox outbreak in African region

15 August 2024

Brazzaville – As the mpox outbreak that has affected the Democratic Republic of the Congo and spread to neighbouring countries continues to grow, World Health Organization (WHO) is intensifying support to countries to scale up measures to curb the virus and save lives.

Read more »

Kenya: Strengthening the health workforce

12 August 2024

Nairobi – Five years ago, Esther Omagwa was one of only two nurses at the Railways Health Centre in Kisumu County, western Kenya. The immense workload often exhausted her, forcing her to turn away clients.

Today, the scene at the community health centre is remarkably different. With the nursing staff now expanded to a team of four, Omagwa and her colleagues are better equipped to deal with the load.

Read more »

African region faces an unprecedent surge in mpox cases

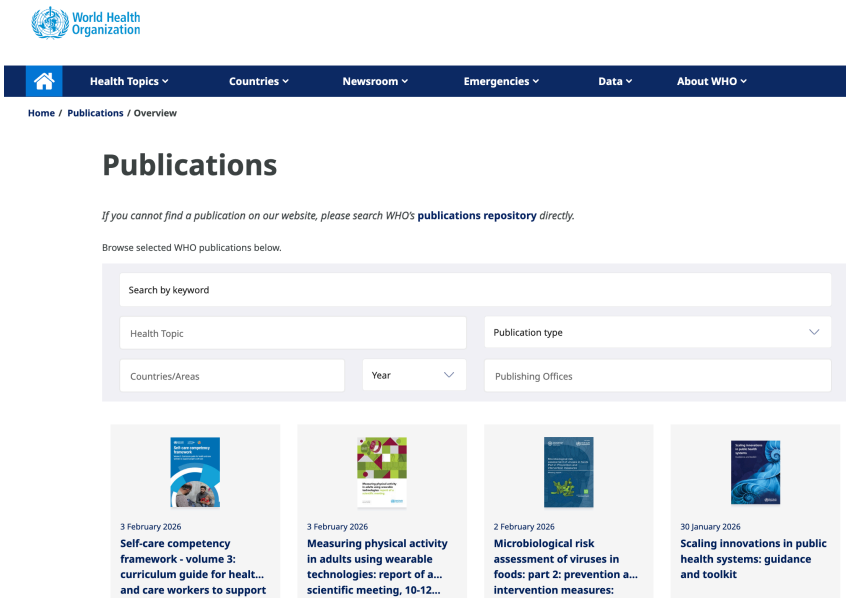
08 August 2024

The African region is experiencing an unprecedented increase in mpox cases since the start of 2024, with more countries previously unaffected by the disease reporting cases in an expanding spread of the virus.

Read more »

WHO Africa News Releases (link)

Example: navigating from a directory with links



Toy example: Wikipedia Crawler

- We will demonstrate coding a crawler with the following toy Wikipedia example

Toy example: Wikipedia Crawler

- We will demonstrate coding a crawler with the following toy Wikipedia example
- Apparently, following the first link in the main text of a Wikipedia article repeatedly will lead to the 'Philosophy' page 97% of the time! (link)
- Let's create a web crawler to test this out, starting from the 'United States' Wikipedia page: https://en.wikipedia.org/wiki/United_States (link)

Toy example: Wikipedia Crawler

- We will demonstrate coding a crawler with the following toy Wikipedia example
- Apparently, following the first link in the main text of a Wikipedia article repeatedly will lead to the 'Philosophy' page 97% of the time! (link)
- Let's create a web crawler to test this out, starting from the 'United States' Wikipedia page: https://en.wikipedia.org/wiki/United_States (link)
- *(Note: in the future, you will likely not be using web crawling in this way. However, we're going to use this as a toy example to illustrate the mechanics of setting up a web crawler. Wikipedia is a good place to do this because they won't block us if we access it too often.)*

Steps We Want our Wikipedia Crawler to Take

- Step 1: Inspect the HTML of the first Wikipedia page we want to scrape

Steps We Want our Wikipedia Crawler to Take

- Step 1: Inspect the HTML of the first Wikipedia page we want to scrape
- Step 2: Parse HTML from first page and extract the first link

Steps We Want our Wikipedia Crawler to Take

- Step 1: Inspect the HTML of the first Wikipedia page we want to scrape
- Step 2: Parse HTML from first page and extract the first link
- Step 3: Follow that link

Steps We Want our Wikipedia Crawler to Take

- Step 1: Inspect the HTML of the first Wikipedia page we want to scrape
- Step 2: Parse HTML from first page and extract the first link
- Step 3: Follow that link
- Step 4: If not on the Philosophy page, repeat the previous step

Steps We Want our Wikipedia Crawler to Take

- Step 1: Inspect the HTML of the first Wikipedia page we want to scrape
- Step 2: Parse HTML from first page and extract the first link
- Step 3: Follow that link
- Step 4: If not on the Philosophy page, repeat the previous step
- Step 5: build in conditions to stop the loop
  - In addition to checking for Philosophy
  - To prevent an infinite loop, we will also stop if if we go to a link we've been to before
  - Or if we've visited 50 links

Step 1: Inspect website and HTML

- Start at https://en.wikipedia.org/wiki/United_States (link)

United States

Article Talk Read View

From Wikipedia, the free encyclopedia

Several terms redirect here. For other uses, see America (disambiguation), US (disambiguation), USA United States of America (disambiguation), and United States (disambiguation).

The **United States of America** (**USA** or **U.S.A.**), commonly known as the **United States** (**US** or **U.S.**) or **America**, is a country primarily located in North America. It is a federal union of 50 states, which also includes its federal capital district of Washington, D.C., and 326 Indian reservations.[j] The 48 contiguous states are bordered by Canada to the north and Mexico to the south. The State of Alaska is non-contiguous and lies to the northwest, while the State of Hawaii is an archipelago in the Pacific Ocean. The United States also asserts sovereignty over five major unincorporated island territories and various uninhabited islands.[k] The country has the world's third-largest land area,[d] second-largest exclusive economic zone, and third-largest population, exceeding 334 million.[l]

United

Flag

Motto
Other

Anthem: "T

- The first in-text link is to “North America”, so we should check for that

Step 1: Inspect website and HTML

- Right click + Inspect...

- The first article link appears to be inside an `<a href>` tag + attribute, inside a `<p>` tag
- Content is only text, with no other nested tag inside

Step 2: parse HTML from first page and extract first link

```
url = "https://en.wikipedia.org/wiki/United_States"
myheader = {
    "User-Agent": "DAP30538CourseBot/1.0 (m.shi@uchicago.edu)"
}
response = requests.get(url, headers = myheader)
soup = BeautifulSoup(response.content, 'lxml')
```

Step 2: extract first link, sped up with AI

AI prompt (link):

I'm using Python with BeautifulSoup to scrape Wikipedia to get the first link on every page. From inspecting the HTML, I see that the link I want: is an <a> tag inside a <p> tag, has an href attribute, contains only text, and has no nested HTML tags inside it.

Step 2: sped up (or slowed down?) with AI

Response:

```
for p in soup.find_all("p"):
    link = p.find(
        lambda tag: tag.name == "a"
        and tag.has_attr("href")
        and tag.string is not None
    )
```

Step 2: sped up (or slowed down?) with AI

Response:

```
for p in soup.find_all("p"):
    link = p.find(
        lambda tag: tag.name == "a"
        and tag.has_attr("href")
        and tag.string is not None
    )
```

Q for AI: what does the `tag.string is not None` syntax do?

Response: returns the tag's text only if the tag contains exactly one text node and no nested elements; otherwise it returns None.

Step 2: sped up (or slowed down?) with AI

Q for AI: I've seen `tag.find(True) is None` before to look for tags with no nested elements. What's different here?

Response: `tag.string is not None` checks that a tag contains exactly one text node only, while `tag.find(True) is None` checks more generally that the tag has no nested HTML tags, regardless of how its text is split.

Step 2: sped up (or slowed down?) with AI

Q for AI: I've seen `tag.find(True) is None` before to look for tags with no nested elements. What's different here?

Response: `tag.string is not None` checks that a tag contains exactly one text node only, while `tag.find(True) is None` checks more generally that the tag has no nested HTML tags, regardless of how its text is split.

Q for AI: Could I have used `tag.find(True) is None` here?

A: Yes, and for your Wikipedia use case, it's actually the better condition. For example, the current condition would have excluded links like `<a href="/wiki/Dog">Big dog</a>` because the text is split internally, meaning it would have been skipped because it has two text nodes.

Step 2: sped up (or slowed down?) with AI

Q for AI: I've seen `tag.find(True) is None` before to look for tags with no nested elements. What's different here?

Response: `tag.string is not None` checks that a tag contains exactly one text node only, while `tag.find(True) is None` checks more generally that the tag has no nested HTML tags, regardless of how its text is split.

Q for AI: Could I have used `tag.find(True) is None` here?

A: Yes, and for your Wikipedia use case, it's actually the better condition. For example, the current condition would have excluded links like `<a href="/wiki/Dog">Big dog</a>` because the text is split internally, meaning it would have been skipped because it has two text nodes.

Takeaway: AI suggested something that likely would have worked at first, but if we didn't probe to understand the syntax that was new to us, debugging that issue down the line would have been very difficult!

Step 2: Parse HTML from first page and extract first link

```
for p in soup.find_all("p"):
    link = p.find(
        lambda tag: tag.name == "a"
        and tag.has_attr("href")
        and tag.find(True) is None
    )

    if link:
        print("First valid link:", link.text)
        print("URL:", link["href"])
        break
```

```
First valid link: North America
URL: /wiki/North_America
```

Step 2: Parse HTML from first page and extract first link

```
for p in soup.find_all("p"):
    link = p.find(
        lambda tag: tag.name == "a"
        and tag.has_attr("href")
        and tag.find(True) is None
    )

    if link:
        print("First valid link:", link.text)
        print("URL:", link["href"])
        break
```

```
First valid link: North America
URL: /wiki/North_America
```

It's close, but it scraped a relative path. We'll have to modify code slightly to make it an absolute path: `'https://en.wikipedia.org/wiki/North_America'`

Step 2: Parse HTML from first page and extract first link

```
p_tag = soup.find_all('p')
for p in p_tag:
    link = p.find(lambda tag: tag.name == "a"
        and tag.has_attr("href")
        and tag.find(True) is None
    )
    if link:
        url = "https://en.wikipedia.org" + link["href"]
        print(url)
        break
```

`https://en.wikipedia.org/wiki/North_America`

Now we can use `url` to crawl to the next page!

Step 3: Follow the first link

To follow the link we collected, our updated `url` becomes the input into a new `request.get()` and call to `BeautifulSoup`

```
# repeating the previous step, but now url has been updated
response = requests.get(url, headers = myheader)
soup = BeautifulSoup(response.content, 'lxml')
```

Step 4: Follow the first link and repeat...

```
p_tag = soup.find_all('p')
for p in p_tag:
    link = p.find(lambda tag: tag.name == "a"
        and tag.has_attr("href")
        and tag.find(True) is None
    )
    if link:
        url = "https://en.wikipedia.org" + link["href"]
        print(url)
        break
```

```
https://en.wikipedia.org/wiki/Continent
```

Back to step 1: inspect website

We should go back to the 'North America' page to confirm that the first link is to 'continent'

North America

Article Talk

From Wikipedia, the free encyclopedia

"North American" redirects here. For other uses, see North American (disambiguation) or Northern United States.

North America is a continent[b] in the Northern and Western Hemispheres.[c] North America is bordered to the north by the Arctic Ocean, to the east by the Atlantic Ocean, to the southeast by South America and the Caribbean Sea, and to the west and south by the Pacific Ocean. The region includes the Bahamas, Bermuda, Canada, the Caribbean, Central America, Clipperton Island, Greenland, Mexico, Saint Pierre and Miquelon, Turks and Caicos Islands, and the United States.

Step 5: Build in conditions to stop the loop

- We have built and tested a few iterations of our crawler. But we want to follow scraping etiquette by limiting the number of requests we make and making sure we don't end up in an infinite loop.
- Preventing infinite loops also
- Conditions to stop the loop:

1. We've hit the Philosophy page (our goal)
2. We've been to this link before (prevents infinite loops)
3. We've visited 50 links (prevents too many requests)

Step 5: Build in conditions to stop the loop

- We have built and tested a few iterations of our crawler. But we want to follow scraping etiquette by limiting the number of requests we make and making sure we don't end up in an infinite loop.
- Preventing infinite loops also
- Conditions to stop the loop:

1. We've hit the Philosophy page (our goal)
2. We've been to this link before (prevents infinite loops)
3. We've visited 50 links (prevents too many requests)

- Let's start with the **third condition**: max out at visiting 50 links

```
for i in range(50):
    # code to crawl here
```

Step 5: Build in conditions to stop the loop

- Then the **first condition**: stop if we've hit the Philosophy page

```
for i in range(50):
    # code to crawl here
    if url == "https://en.wikipedia.org/wiki/Philosophy":
        print('Ended up at Philosophy in ' + str(i) + ' tries!')
        break
```

- `break` exits the `for` loop as soon as the condition is triggered

Step 5: Build in conditions to stop the loop

- Then we can tackle the **second condition**: exit if we've visited this link before

```
visited_urls = []
for i in range(50):
    if url in visited_urls:
        print('Stopped because ended up in a loop.')
        break
    # code to crawl here
    if url == "https://en.wikipedia.org/wiki/Philosophy":
        print('Ended up at Philosophy in ' + str(i) + ' tries!')
        break
    visited_urls.append(url)
```

- Initialize an empty list: `visited_urls[]`
- `break` if the url we extract has already been visited
- If none of the conditions to `break` are true, then add this url to `visited_urls`

Bringing it all together

```
url = "https://en.wikipedia.org/wiki/United_States"
visited_urls = []
for i in range(50): # cond 3: stop at 50 loops
    # cond 2: stop if visited before
    if url in visited_urls:
        print('Stopped because ended up in a loop.')
        break
    visited_urls.append(url)

    # code to crawl
    response = requests.get(url, headers = myheader)
    soup = BeautifulSoup(response.content, 'lxml')
    p_tag = soup.find_all('p')
    for p in p_tag:
        link = p.find(lambda tag: tag.name == "a"
            and tag.has_attr("href")
            and tag.string is not None
        )
        if link:
            url = "https://en.wikipedia.org" + link["href"]
            print(url)
            break

    # cond 1: stop if at philosophy
    if url == "https://en.wikipedia.org/wiki/Philosophy":
        print('Ended up at Philosophy in ' + str(i) + ' tries!')
        break
```

Bringing it all together

```
https://en.wikipedia.org/wiki/North_America
https://en.wikipedia.org/wiki/Continent
https://en.wikipedia.org/wiki/Convention_(norm)
https://en.wikipedia.org/wiki/Social_norm
https://en.wikipedia.org/wiki/Acceptance
https://en.wikipedia.org/wiki/Psychology
https://en.wikipedia.org/wiki/Mind
https://en.wikipedia.org/wiki/Thought
https://en.wikipedia.org/wiki/Cognition
https://en.wikipedia.org/wiki/Knowledge
https://en.wikipedia.org/wiki/Declarative_knowledge
https://en.wikipedia.org/wiki/Awareness
https://en.wikipedia.org/wiki/Philosophy
Ended up at Philosophy in 12 tries!
```

Web Crawler: Summary

- We can go beyond scraping from individual URLs by crawling
- Crawler pulls in URLs stored in a tags
  - But you may need a more specific `.find_all()` than just this
  - Use AI to develop your search conditions - but always check that you understand it work!
- Crawling requires using loops, so we have to design the crawler carefully to avoid infinite loops

Web Scraping: Summary

- Web scraping allows you to systematically access and download text and data from websites using Python
- To scrape HTML-based sites:
  - Use `requests` + `BeautifulSoup` packages
  - Usually the first version of your scraper will require tweaking: manual inspection + AI assistance
- Scraping links enables you to *web crawl*
- High up-front cost to developing a scraper, so exhaust other tools (e.g., direct download, API) before committing to it