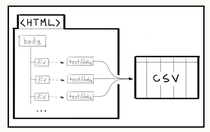
A Simple Cheat Sheet for Web Scraping with Python

last updated July 2023

What is web scraping?

Web scraping is the process of extracting data from the internet programmatically

It's used when you want to obtain a large amount of data from a website that doesn't provide a method to download that data.



Make sure to check the website's policy on web scraping before any scraping. Generally, try to avoid spamming a site with too many/too frequent requests.

What tools do you need?

There are multiple options for web scraping in Python, it ultimately depends on the task.

1. First, you need an **IDE (Integrated development environment)**:

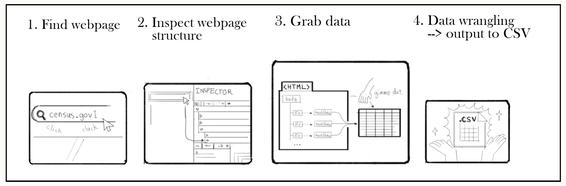
   I prefer using an **.ipynb file** in **Jupyter Notebook** or **VS Code**
   (IMO, using cells fits the iterative, incremental nature of web scraping)

2. You will also need to install and import **scraping packages**:

   beautifulsoup4 (& requests), **selenium**, **pandas** (yes pandas can be used for scraping)

   by running `pip install package_name` in the command prompt at your project directory

What does a typical workflow look like?



Option 0: **Pandas**

If the data is present in a neat **<table>** element → always use Pandas' **pd.read_html()** (it's simply the simplest option)

Head

```
import pandas as pd
tables = pd.read_html(url, match='matching string') # returns a list of dataframes
                                                    # created from each table in the html
df = tables[1]                                      # to grab a specific table, grab via index
df.head()                                           # preview table
```

source: pandas.read_html

Option 1: **BeautifulSoup**

Docs: **Beautiful Soup Documentation**

Use BeautifulSoup when working with a *static* webpage
(i.e. when the desired data is entirely within the page's HTML representation)

0. Imports + Setup

```
import requests
import BeautifulSoup from bs4

url = 'example.com'
request = requests.get(url)                          # grab the HTML file from the url
soup = BeautifulSoup(request.content, 'html.parser') # create soup object from HTML file and parser
```

1. Getting Elements

```
# common methods
first_elt = soup.find('a')                           # finds the first <a> tag

multiple_elts = soup.find_all('div',class_='c-name') # finds all <div> tags with class 'c-name'
# soup.find_all('div','c-name')                      # can leave out class_, since class is 2nd arg
# soup.find_all('div',{class:'c-name', id:'i-name'}) # or specify multiple identifiers with dictionary
# soup('div')                                        # same thing as calling soup.find_all('div')

soup.find('div',id_='id-name')                       # finds <div> tag with id 'id-name'

soup.find(string="Exact text")                       # finds by exact text

---

# less common methods
soup.find_all('div', id=lambda x: x and x.startswith('start-')) # finds <div> tags with ids that
                                                                # start with 'start-'
import re
soup.find(string=re.compile("Link")).parent          # finds first element with partial text 'Link'

soup.find('div',class_='c-name1 c-name2')            # finds element with multiple classes
soup.find('div',class_=['c-name1', 'c-name2'])       # or use list when multi-class
soup.select("div.c-name1.c-name2")                   # or .select()

first_elt.next_sibling()                             # grabs the next child with the same parent as
                                                     # first_elt, (the next sibling); useful when an
                                                     # elt doesn't have identifier, but first_elt does

---

# I almost never use these
children = first_elt.contents                        # returns direct children of first_elt in a list
all_children = first_elt.descendants                 # returns all children, down to leaf node
papa = first_elt.parent                              # returns direct parent
ancestors = first_elt.parents                        # returns all parents, up to root node
```

Note: you don't need to list *all* classes to find a multi-class element, but being more specific can help narrow the list down

2. Getting Attributes

```
first_href = e.get('href')              # gets the value of the href attribute
# e['href']                             # can also index with []

all_hrefs = [e.get('href') for e in el] # use list comprehension to return list of
                                        # the hrefs from multiple_elts

text = first_elt.get_text()             # gets text of first_elt
```

3. Misc Tips

```
print(first_href.prettify())                    # prints element cleanly

soup.html.find_all("title", recursive=False)    # recursive=False only finds direct children

element.encode("utf-8")                         # encodes tag/soup objects as other encodings

tag.name = "blockquote"                        # you can assign/re-assign the name of a tag
tag['id'] = "new-id"                           # same with tag attributes
                                               # (though I haven't discovered a use for this)

# use soup.css.select() or soup.select() to use css-selectors
```

more on css.select: Beautiful Soup Documentation – css-selectors

Option 2: **Selenium**

Docs: **Selenium with Python**

Web scraping with selenium involves using *web automation* to collect data through a *web driver* rather than working with a local HTML file as beautiful soup does
Use Selenium when:

- the data requires navigation (querying, clicking buttons, etc.) to access
- the data is located across multiple pages with non-uniform URLs,
- or the elements don't have good class/id naming conventions (i.e. XPATH is the best option)

0. Imports + Setup

```
from selenium import webdriver
from selenium.webdriver.edge.service import Service
from webdriver_manager.microsoft import EdgeChromiumDriverManager

# download the most updated version of the driver, open the driver, and assign to driver variable
driver = webdriver.Edge(service=Service(EdgeChromiumDriverManager().install()))

# driver = webdriver.Edge(path/to/driver/executable)   # can also download web driver locally and
                                                       # provide it's path (make sure it's the
                                                       # newest version!)
url = 'example.com'
driver.get(url)                                        # opens driver with the specified URL
```

If you prefer other browsers, there are also Chrome, Firefox, and Safari web drivers

0b. Managing Driver + Driver Options

```
driver.quit()                                          # closes entire driver (all tabs)
driver.close()                                         # closes currently focused driver tab

driver.navigate().back()                               # returns to previous page in browser history

driver.execute_script("window.open('');")              # opens a new tab
driver.switch_to.window(driver.window_handles[1]) # switch to new tab (2nd tab) assuming 1 tab was open
driver.get(new_url)                                    # get new_url on new tab

alert = driver.switch_to.alert
alert.accept()                                         # closes popups

from selenium.webdriver.edge.options import Options
options = Options()
options.add_argument("--incognito")                    # add new driver option
driver = webdriver.Chrome(options=options)             # passes options into driver instance
# there are many more driver options like window size, headers, gpu, etc...
```

1. Getting Element(s)

```
from selenium.webdriver.common.by import By            # for grabbing elements by attribute
# common methods for getting element(s)
first_elt = driver.find_element(By.ID, 'id-name')      # finds the first element with id 'id-name'

multiple_elts = driver.find_elements(By.CLASS, 'c-name') # finds all elements with class 'c-name'
                                                         # note we're using .find_elements() instead

xpath_elt = driver.find_element(By.XPATH,'//xpath/elt')  # use as last resort when no suitable class/id
                                                         # identifier exists
                                                         # should be last resort because small change
                                                         # in HTML can invalidate XPATH

first_elt.find_elements(By.CLASS, ...)                   # can also call .find_elements() on other
                                                         # elements (this searches first_elt's children)

# other less common methods include: TAG_NAME, LINK_TEXT, PARTIAL_LINK_TEXT, NAME, CSS_SELECTOR
```

read more here: 4. Locating Elements – Selenium Python

To find the XPATH of an element: 1) open inspector (Ctrl+Shift+I) or right click –> Inspect
2) select element
3) right click the element in inspector
4) copy XPATH

1b. XPATH examples

```
driver.find_element(By.XPATH,"//div[@class='gallery'][1]/img[@id='some-image']")

//                <-- looks for all elements from the root node
/                 <-- looks for first elt from root node (starting with / means absolute path)
[@attr]           <-- looks for elements with this attribute
[@attr=value]     <-- looks for elements with attribute equal to value
/div[1]           <-- selects the first (XPath uses 1-indexing) div from the root node
/div[position()<3] <-- selects the first 3 divs from the root node
/div[1]/*         <-- wildcard, matches any child element of the 1st div
```

source: XPath Syntax (w3schools.com)

more info on XPATH syntax

1c. Getting Attributes from Elements

```
elt_href = first_elt.get_attribute('href')             # gets the href attribute value from first_elt

hrefs = [e.get_attribute('href') for e in multiple_elts] # use list comprehension to return
                                                         # a list of the hrefs from multiple_elts

text = first_elt.text                                  # gets the inner text of first_elt
                                                       # e.g. <div><p>this is the inner text</p></div>

# for dynamic inputs (like if a checkbox is checked) use get_property() instead of get_attribute()
checkbox.get_property('checked')
```

2. Waits

```
# Sometimes, the web page takes a while to load a page/element, so we need to tell the driver to
# WAIT until the element (or another reference element is loaded) to grab your element
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

time_wait = 10
waited_elt = WebDriverWait(driver, time_wait).until(
    EC.presence_of_element_located((By.ID, "elt-id")) # waits 10 seconds before looking for this elt
)

# driver.implicitly_wait(time_wait)                  # can also set wait for all future elements
# Note: time.sleep(sec) is another alternative to WebDriverWait, but a more hacky option
```

read more here: 5. Waits – Selenium Python

3. Input

```
from selenium.webdriver.common.keys import Keys  # includes special keys like Ctrl, Backspace, etc.
from selenium.webdriver.support.ui import Select # for dropdowns

btn.click()                                      # clicks on a button, works for checkbox/radio

dropdown = Select(driver.find_element_by_id('dropdown')) # Select option from a dropdown
dropdown.select_by_visible_text('Option 1')      # by text
# select by value select.select_by_value('1')    # by value attribute

text_input.clear()                               # clear text input, type something, then submit
text_input.send_keys('example input')
text_input.submit()
# text_input.send_keys(Keys.RETURN)              # can also press enter to trigger form submit

# more advanced actions require importing ActionChains
from selenium.webdriver.common.action_chains import ActionChains
actions = ActionChains(driver)                   # create ActionChains instance

driver.execute_script("arguments[0].scrollIntoView();", elt) # scrolls till element is in view
elt.location_once_scrolled_into_view             # scrolls to element + returns coords.

actions.move_to_element(elt).perform()           # moves mouse to element (triggers hover)

actions.double_click(elt)                        # double clicks on element

# a much more comprehensive list of Actions, Keys, etc. can be found in the full API ↓
```

full list of actions in API: 7. WebDriver API

good option for dealing with hidden submenu items:
first .move_to_element(submenu), then .click(submenu_item)

4. Misc Tips

```
from selenium.common.exceptions import [TheNameOfTheExceptionClass] # selenium has exception handling

driver.maximize_window()             # you can probably guess what this does

driver.save_screenshot("image.png") # takes screenshot of element

"//*[contains(@id,'_ending')]"       # XPATH can match partial identifiers
"//*[ends-with(@href,'.gov')]"       # works with attributes as well

driver.switch_to_frame("f-name")     # accesses iframes by name or ID
```


Created by: **Tyler Wu** Find more cheat sheets at: **The Pig Pencil**