This blog is about approaches I naturally used with R's `rvest` package and Python's `BeautifulSoup` library.

Here are two versions of code which I use to scrape all the breifings

This webscraper extracts:

1) Date of the Breifing
2) The title of the Breifing
3) The URL to the Breifing
4) The The Issue Type

and puts them in a data frame.

The differences between the way I did this in Python vs R:

Python

(a) I grabbed the data using the xml
(b) Parsing the data was done with the html classes (and cleaned with a small amount of Regex)
(c) I used `for` loops
(d) I had to import other libraries besides for bs4

R

(a) I used a CSS selector to get the raw data.
(b) The data was parsed using good ol' regular expressions.
(c) I used `sapply()`
(d) I just used rvest and the base library.

This is a comparison between how I learned to webscrape in Python vs How I learned how to do it in R. Lets jump in and see which one did faster!

Python Version with BeautifulSoup

```
# A simple webscraper providing a dataset of all Whitehouse Breifings
from bs4 import BeautifulSoup
import requests
import pandas as pd
import re
import lxml


def get_whitehouse_breifings():
    # Generalize to all pages

    orig_link = requests.get("https://www.whitehouse.gov/briefings-statements/")

    orig_content = orig_link.content

    sp = BeautifulSoup(orig_content, 'lxml')

    pages = sp.find_all('a', {'class': 'page-numbers'})

    the_pages = []
    for pg in pages:
        the_pages.append(pg.get_text())

    # Now make set of links

    the_links = []

    for num in range(1, int(max(the_pages)) + 1):
        the_links.append('https://www.whitehouse.gov/briefings-statements/' + 'page/' +
str(num) + '/')

    dat = pd.DataFrame()
    for link in the_links:
        link_content = requests.get(link)
        link_content = link_content.content
        sp = BeautifulSoup(link_content, 'lxml')
        h2_links = sp.find_all('h2')
        date_links = sp.find_all('p', {"class": "meta__date"})
        breif_links = sp.find_all('div', {"class": "briefing-
statement__content"})

        title = []
        urls = []
        date = []
        breifing_type = []
        for i in h2_links:
            a_tag = i.find('a')
            urls.append(a_tag.attrs['href'])
            title.append(a_tag.get_text())
        for j in date_links:
            d_tag = j.find('time')
            date.append(d_tag.get_text())
        for k in breif_links:
            b_tag = k.find('p')
            b_tag = b_tag.get_text()
            b_tag = re.sub('\\t', '', b_tag)
            b_tag = re.sub('\\n', '', b_tag)
            breifing_type.append(b_tag)

        dt = pd.DataFrame(list(zip(date, title, urls, breifing_type)))

        dat = pd.concat([dat, dt])

    dat.rename(columns={"Date": date, "Title": title, "URL": urls, "Issue Type":
breifing_type})
    return (dat)
```

Running the code, Python’s Time

```
import time
start_time=time.time()

pdt = get_whitehouse_breifings()


# Time taken to run code
print("--- %s seconds ---" % (time.time() - start_time))
```

```
## --- 162.8423991203308 seconds ---
```

R Version with rvest

```
library(rvest)

get_whitehouse_breifings<- function(){
  #Preliminary Functions




  pipeit<-function(url,code){
    read_html(url)%>%html_nodes(code)%>%html_text()
  }

  pipelink<-function(url,code){
    read_html(url)%>%html_nodes(code)%>%html_attr("href")
  }


  first_link<-"https://www.whitehouse.gov/briefings-statements/"

  # Get total number of pages

  pages<-pipeit(first_link,".page-numbers")

  pages<-as.numeric(pages[length(pages)])

  #Get all links
  all_pages<-c()

  for (i in 1:pages){
    all_pages[i]<-paste0(first_link,"page/",i,"/")
  }



  urls<-unname(sapply(all_pages,function(x){
        pipelink(x,".briefing-statement__title a")
        })) %>% unlist()

  breifing_content<-unname(sapply(all_pages,function(x){
    pipeit(x,".briefing-statement__content")
  })) %>%  unlist()


  # Data Wrangling

  test<-unname(sapply(breifing_content,function(x) gsub("\\n|\\t","_",x)))

  test<-unname(sapply(test,function(x) strsplit(x,"_")))

  test<-unname(sapply(test,function(x) x[x!=""]))
  breifing_type<-unname(sapply(test,function(x) x[1])) %>% unlist()
  title<-unname(sapply(test,function(x) x[2])) %>% unlist()
  dat<-unname(sapply(test,function(x) x[length(x)])) %>% unlist()


  dt<- data.frame("Date"=dat,"Title"=title,"URL"=urls,"Issue Type"=
breifing_type)

  dt
}
```

Running the code,R's Time

```
##    user  system elapsed
##   16.77    4.22  415.95
```

Analysis and Conclusion:

On my machine Python was **waaaaay** faster than R. This was primarily because the function I wrote in R had to go over the website a second time to extract links. Could it be sped up if I wrote the code extracting text and links in one step? Very likely. But I would have to change the approach to be similar to how I did it in Python.

For me `rvest` seems to be great for "quick and dirty" code (Point and click with a CSS selector, put it in a function, iterate accross pages; Repeat for next field). `BeautifulSoup` seems like its better for more methodical scraping. The approach is naturally more html heavy.

Python requires one to refrence the library every time they call a function from it, which for myself being a native R user find frustrating as opposed to just attaching the library to the script.

For R you have to play with the data structure (from lists to vectors) to get the data to be coerced to a dataframe. I didn't need to do any of this for Python.

I'm sure theres more to write about these libraries (and how there are better ways to do it in both of these languages), but I'm happy that I am aquainted with them both!