Python for journalists: redux

Paul Bradshaw

This week:

Recapping the first 4 weeks

- Variables: strings, numbers, Booleans, etc.
- Questioning data: pandas
- Scraping data: lists, dicts and BeautifulSoup

Why oh why?

- You need to store info: **variables**
- You need to repeat actions: ***for* loops & lists**
- You need to ask questions: **functions**
- You need to perform related actions: **libraries**

More variables in Python: text and other beasts

In a previous notebook we looked at how to create variables in Python and perform a calculation with those.

Those variables all happened to be numbers. We're going to need them again so let's recreate them in this notebook first:

In [ ]:

```
#store the numbers of requests in a new variable called 'fcorequests'
fcorequests = 48
#store the numbers of refusals in a new variable called 'sec27refusals'
sec27refusals = 28
#calculate a percentage by dividing the part (refusals) by the whole (requests)
#and store in a new variable called 'percrefused'
percrefused = sec27refusals/fcorequests
#print it
print(percrefused)
#multiply by 100 to make it easier to 'read' as a percentage
print(percrefused*100)
```

```
0.5833333333333334
58.333333333333336
```

(A quick recap: these figures are taken from Freedom of Information statistics: April to June 2021 bulletin - the data tables link is here and go to the sheet called '10_Exemptions'.)

Creating text variables: 'strings'

Data doesn't just come in the form of numbers, however. It's almost certain that at some point we're going to need to store some **text**.

For example, we might need to store the recipients of money being spent, in order to calculate which one received the most spending.

In our FOI story we might want to store the names of organisations, or the sections of the FOI Act that are being used as the basis for refusals, or a note from the spreadsheet.

Lists, dictionaries - and functions

In this notebook we look at two more common types of variables: **lists** and **dictionaries**, how to recognise them, how to create them, and what you might use them for as a journalist.

Lists are like columns of data

A list is a type of variable that allows you to store *multiple* values. It might be a list of numbers, a list of strings, a list of `True/False` values, or a mix of those.

You can even have a list of lists, but that's a bit too mind-bending to get into now.

To create a list, you need to put **square brackets** around your list of items, and **separate each one with a comma**, like so:

In [ ]:
```
#this is a list of numbers
refusals = [1, 11, 4, 0]
#this is a list of strings
orgs = ["AGO", "Cabinet Office", "DBEIS", "DCMS"]
#this is a list of Booleans
dept_tf = [False, False, True, True]
```

Those square brackets will also show when a list is printed - so it's a key way to recognise that you're dealing with a list (which might be the case if you've imported some data and then extracted one column).

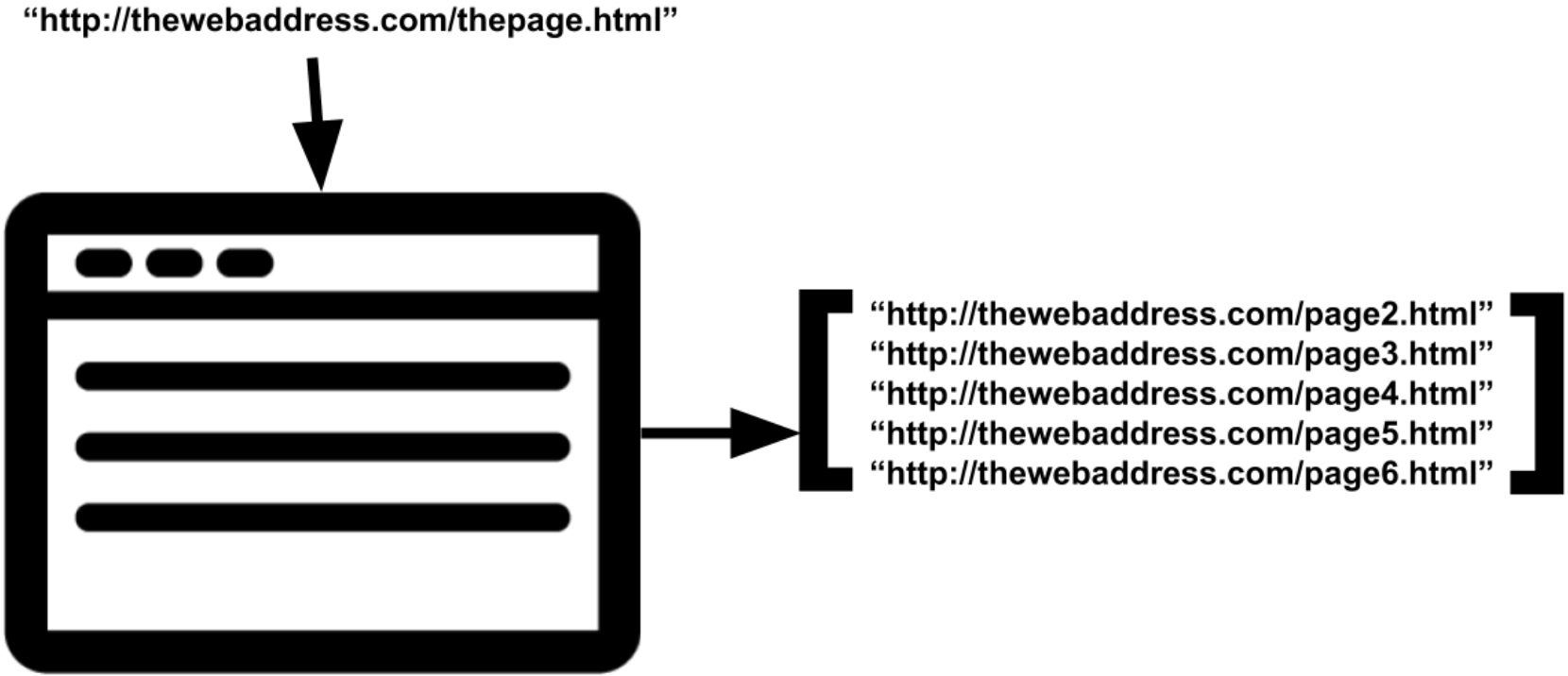
In [ ]:
```
print(dept_tf)
```

```
[False, False, True, True]
```

The data in the first two lists is from the Freedom of Information statistics: April to June 2021 bulletin - specifically the data tables in the sheet called '10_Exemptions'.

Lists

- A list starts and ends with square brackets
- `mydata = [“Kim”, “Jim”, “Tim”]`
- For storing multiple items
- Can be list of strings, integers, etc.



Go to a page -> scrape HTML -> extract links from HTML

Lists are **for** looping!

- Loop through a list using **for**

```
for i in mylist:
```

- The ‘i’ is a name you are giving each item
- The colon introduces intended commands

Dictionaries (‘dicts’)

- A dictionary starts and ends with curly brackets
- Contains a **list of pairs**, joined by a colon
- `mydict = {“dept” : “DfE”, “fois”:5}`
- Called ‘key-value’ pairs because used for storing labels (keys) and data (values)

Using libraries in Python to access more functions

All of the functions in previous notebooks are built into Python. But you can access thousands more functions from **libraries** that people have created (sometimes called **modules**).

A **library** is a collection of functions and other code that someone has created - typically to solve a particular problem. For example:

- The `pandas` library was created to solve problems relating to data analysis
- The `matplotlib` library was created to add extra visualisation functionality to Python
- The `scraperwiki` library was created to solve scraping problems. It's not the only one: Beautiful Soup is another library for this, too.
- The `pdf2xml` library was designed to help deal with PDFs in Python (by converting them to xml)
- The `re` library provides functions to use **regular expressions** in Python, in order to describe patterns you might want to match (and fetch) in text or webpages.
- The `os` library was designed to add "operating system dependent functionality" such as reading or writing files.
- The `Scikit-learn` library allows you to use machine learning in Python
- The `NumPy` library has lots of functions to do more advanced mathematical processing

A good principle to bear in mind when using programming languages is that if you come across a problem, chances are that someone else has already come across a similar problem - and create a library to solve it. A bit of googling around can help you find that, and learn how to use it to help solve your problem.

To use a library's functions you need to **import** it.

This is done by using the conveniently-named `import` function, followed by the name of the library.

In [ ]:
```
#import the pandas library to use its functions
import pandas
```

First, break down the problem(s)

- E.g. Fetch a page from a URL
- Drill down into that page
- Extract text/links/etc.
- Store them in a dataframe
- Export them as a CSV

Find libraries to solve problems

- Fetch a page from a URL: **requests**
- Drill down into that page: **BeautifulSoup**
- Extract text/links/etc.: **BeautifulSoup**
- Store them in a dataframe: **pandas**
- Export them as a CSV: **pandas**

The pandas library

- `import pandas as pd`
- Functions for handling/analysing data
- Uses a **dataframe** to hold data
- `mydf = pd.DataFrame( { DATA HERE } )`
- `pd.from_csv("URLGOESHERE")`
- `mydf.to_csv("data.csv")`

Intro to `pandas`

This notebook introduces basic techniques in using `pandas` for data analysis.

First, we need to import the `pandas` library. This is pre-installed in Colab notebooks, so doesn't need installing - it only needs bringing in with the `import` command.

It's also quite common to rename the library when it's imported, as `pd`, like so:

In [ ]:
```
import pandas as pd
```

The inverted pyramid of data journalism outlines 5 stages:

1. Compile
2. Clean
3. Combine
4. Context
5. Clean

And running throughout it: **question**.

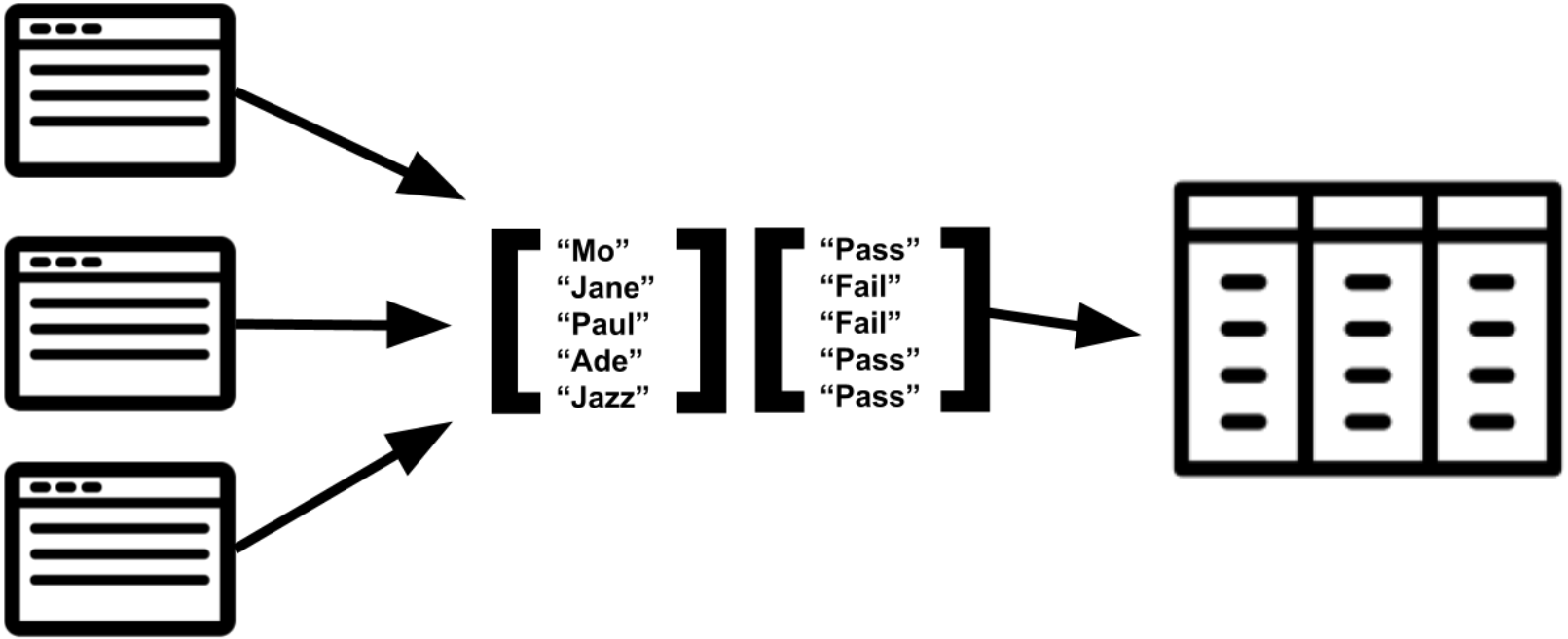
Let's start with compiling in `pandas`.

Compiling data: importing a CSV

The easiest way to compile data in a Colab notebook is to upload the data to the Files area on the left hand side of Colab. Once in the Files view, it can be brought into the notebook with the `read_csv()` function.

Colab already has a 'sample_data' folder in Files with 4 CSV files and a JSON file. We can export one of those to demonstrate:

In [ ]:
```
#import the CSV from the Files in Colab
```



Loop through each page -> scrape HTML -> extract specific info from HTML -> add row to table -> download

The requests library

- Fetches a document from a URL

```
page = requests.get("https://www.bbc.co.uk/news")
```

The BeautifulSoup library

- Converts a webpage into a ‘parsable’ object

```
soup = BeautifulSoup(page.content)
```

Using .select( ) (CSS selectors)

```
h2s = soup.select('h2[class="gel-double-pica-bold"]')
```

(Select any tags matching the description)

```
#grab the h2 tags with the specified class
h2s = soup.select('h2[class="gel-double-pica-bold"]')
#create an empty list
headlines = []
#loop through the h2 matches
for i in h2s:
  #extract the text for that h2
  h2text = i.get_text()
  #add it to the list
  headlines.append(h2text)
```

Loop or use index to access item

Grab the item index 0 in that list of matches

```
h2s[0]
```

Loop through and print each item

```
for i in h2s:
    print(i)
```

Extract text or links

- **i.get_text( )** - grab the text contents of the targeted tags
- **i[‘href’]** - grab the URL that’s linked to (the href= value)

Saving the results: pandas

- Create a dataframe using **pd.DataFrame( )**

```
mydf = pd.DataFrame({"names":list1,
                     "scores":list2})
```

- Brackets include: a dictionary
- Key(s) before colon are your column heading(s)
- Value(s) after colon are your list(s)

```
#create two lists
list1 = ['Paul','Maeve','Alice']
list2 = [20,30,40]
#create a dataframe
simpledf =
pd.DataFrame({"names":list1,
              "scores":list2})
```

```
h2s = soup.select('h2[class="gel-double-pica-bold"]')
headlines = []

for i in h2s:
  h2text = i.get_text()
  headlines.append(h2text)

#create a dataframe with that list
bbcheadlines = pd.DataFrame({"headlines": headlines})
```

Reuse the code.

All you have to change are:

1. The URL (you want scraped)
2. The selectors (for that page)
3. Extra lists for extra columns (repeat & adapt existing code)

Scraping multiple pages

- Once you've scraped one page, you can store the 'steps' in your own **function**
- **Loop** through a list of URLs and apply that function to each
- **Append** the results to a dataframe as you go

```
#define a function
def scrapepage(theurl):
  #fetch the page from the URL
  page = requests.get(theurl)
  #parse the page into a 'soup' object
  soup = BeautifulSoup(page.content, 'html.parser')
  #grab all the headlines - we've identified a class attribute they all have
  headlines = soup.select('div a[class="gs-c-promo-heading gs-o-faux-block-link__overlay-link gel-pica-bold nw-o-link-split__anchor"]')
  #create two empty lists for the two pieces of information we want to extract
  headlinetext = []
  links = []
  #loop through them
  for i in headlines:
    #extract the text
    headtext = i.get_text()
    #extract the link
    headlink = i['href']
    #add the text to the previously empty list
    headlinetext.append(headtext)
    #add the link to a second empty list
    links.append(headlink)
  #check that both are the same length
  print(len(headlinetext))
  print(len(links))
  #create a dataframe to store them
  df = pd.DataFrame({"headline" : headlinetext, "link" : links})
  return(df)
```

https://github.com/paulbradshaw/pythonin12parts/blob/main/part4/08createFunctionsScraperBS.ipynb

```
#Create a dataframe to store the data we are about to scrape
allresults = pandas.DataFrame()

#create a list of URLs to scrape
urllist = ["https://www.bbc.co.uk/news/world","https://www.bbc.co.uk/news/health", "https://www.bbc.co.uk/news/entertainment_and_arts"]
```

```
#then loop through them and add to the URL
for i in urllist:
  #scrape that url
  df = scrapepage(i)
  print(df)
  #add the new data frame to the existing data frame
  allresults = allresults.append(df)

#print final dataframe
print(allresults)
```

https://github.com/paulbradshaw/pythonin12parts/blob/main/part4/08createFunctionsScraperBS.ipynb

Key points

- Use **template code** to fetch & scrape a URL
- Change the **URL**, change the **selectors** for your target page(s), & repeat — that's it
- Use **append** to add to empty list or dataframe