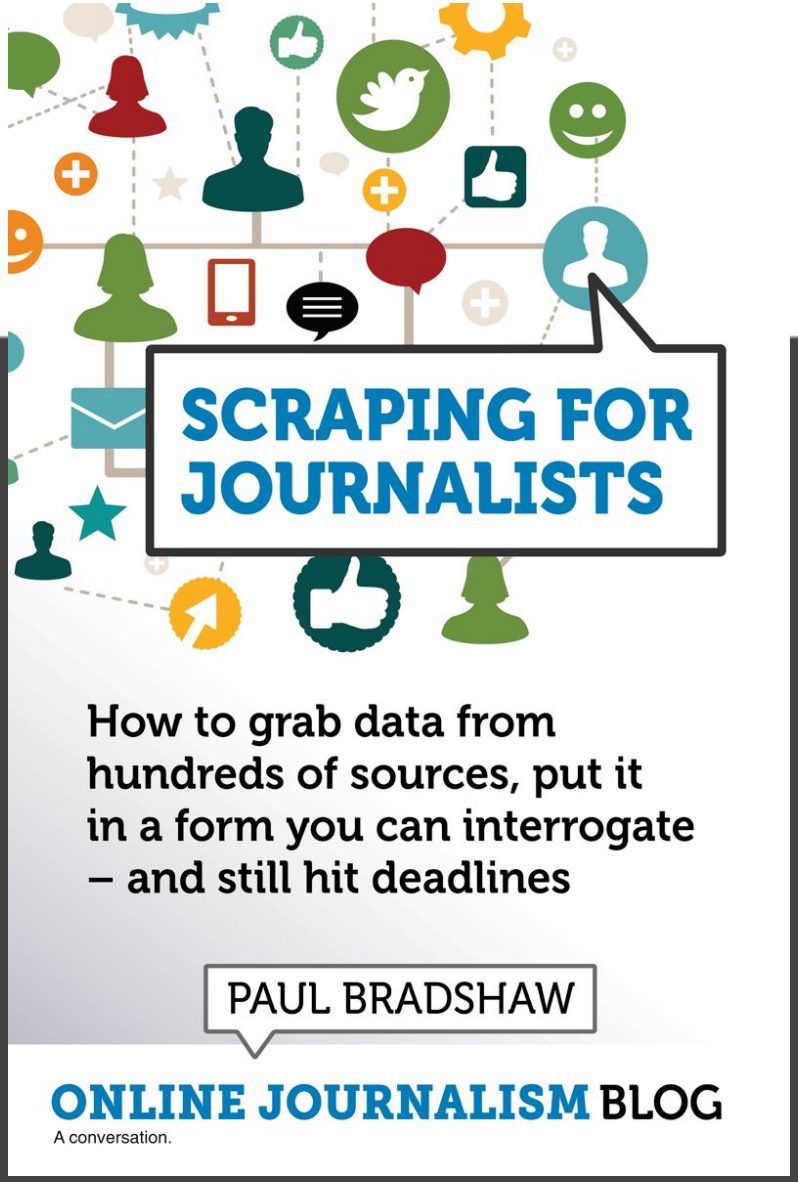
Creating functions for scrapers



Paul Bradshaw

Leanpub.com/scrapingforjournalists

What we'll cover

- How to create your own functions
- Scraping multiple pages

```
def sayhello():
  print("hello")
```

Define a function

Name it

Name any ingredients (parameters) in parentheses

Don’t forget the colon!

Indented lines of code will run when the function is ‘called’

Those ingredients

- Start with **`def`**
- Then name the function (arbitrary)
- Then brackets
- Inside those: name the ingredients
- Then colon
- Then indented lines which represent the ‘recipe’ you are storing in the function (this will likely use the ingredients you named)

```
def print_this_word(thisword):
  print(thisword)
```

‘Calling’ the function

...is like using any other function:

- Type the name of the function
- Then brackets
- Inside those: specify the ingredient(s) (‘arguments’)
- Run it!

```
print_this_word("pumpkin")
```

When this function is called
it needs one ingredient.
We ‘pass’ that inside the parentheses

If a function has multiple ingredients they are separated by commas

```
def addtwonumbers(numone, numtwo):
  #add the two ingredients
  total = numone+numtwo
  #return that value
  return(total)
```

The `return` command is often used to return information to whatever 'called' the function

```
#call the function and
#store result in a variable
whatisit = addtwonumbers(3,8)
print(whatisit)
```

This function needs two ingredients, so we 'pass' those with commas between

The results 'returned' by the function are stored in a new variable

```
#define a function

def scrapepage(theurl):

  print("scraping", theurl)

  #scrape the webpage at that url and store in 'html'

  html = scraperwiki.scrape(theurl, user_agent="Mozilla/5.0
(Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/68.0.3440.106 Safari/537.36")

  #convert 'html' into an lxml object so we can drill into it

  root = lxml.html.fromstring(html)

  ...

  #return the data frame to whatever called the function

  return(df)
```

You’ve already written the code!

Before:

```
[ ] #store the url we want to scrape
    theurl = "https://www.nhs.uk/service-search/other-services/Eating-disorders/N
    #scrape the webpage at that url and store in 'html'
    #without a user agent we get a 403 error on this webpage
    #see https://github.com/sensiblecodeio/scraperwiki-python for documentation
    html = scraperwiki.scrape(theurl, user_agent="Mozilla/5.0 (Macintosh; Intel M
    #convert 'html' into an lxml object so we can drill into it
    root = lxml.html.fromstring(html)
```

After:

```
[ ] #define a function
    def scrapepage(theurl):
      print("scraping", theurl)
      #scrape the webpage at that url and store in 'html'
      html = scraperwiki.scrape(theurl, user_agent="Mozilla/5.0 (Ma
      #convert 'html' into an lxml object so we can drill into it
      root = lxml.html.fromstring(html)
      #grab the contents of every <th> tag
      servicenames = root.cssselect('th')
```

Adapting your code

- Instead of a specific URL string, you use a **variable** to represent ‘any url’
- There may be code to handle **variation** between URLs (e.g. different numbers of items)
- Add a line to **‘return’** the results once the scraper function is finished

Running a function on multiple URLs (lists again!)

```
#Create a dataframe to store the data we are about to scrape

dfhere = pandas.DataFrame(columns=["servicename","tel"])


#first, store the URL up to the page number

firsturlpart =
"https://www.nhs.uk/service-search/other-services/Eating-disorders/No
ttingham/Results/102/-1.158/52.955/1797/15942?distance=500&ResultsOnP
ageValue=100&isNational=0&totalItems=805&currentPage="

#next create a list of page numbers from 1 to 9

pagelist = range(1,10)

#then loop through them and add to the URL

for i in pagelist:

  ...
```

```
for i in pagelist:
  #convert number to string so it can be combined with URL
  pagenumberasstring = str(i)
  #combine that with URL
  pageurl = firsturlpart+pagenumberasstring
  #scrape the page and store results in df
  df = scrapepage(pageurl)
  print(df)
  #add the new data frame to the existing data frame
  dfhere = dfhere.append(df)
  print(dfhere)
```

What’s happening

- We create an empty data frame for the results of the scraper
- We loop through the URLs we want to scrape, and run the scraper function on each one
- Each time it stores the data frame ‘returned’ by the function in a variable
- We then update the empty data frame by appending this new data frame to it
- After 1 loop it has 100 items, after 2 it has 200 (100+100 more) and so on

Handling variation (the last page)

```
#count how many items there are - subtracting 3 for the 3 extra
results we don't want

items = len(servicenames)-3

#set a limit to those items, normally the last 100, but less on the
last page

servicenames = servicenames[-items:]
```

Recap

- Want it done more than once? Create a user-defined function

```
def iamlazy(ingredient1, ingredient2):
   dosomething(ingredient1)
```

- The function turns your previous code into a recipe that can be run on multiple URLs
- Trial and error: later pages may not be quite the same - adapt code to handle errors

Try it now:

- Create a notebook and put the code you've already written for one page into a function
- Test it on the same page - does it work?
- Test it on a couple pages
- Test it on the last page