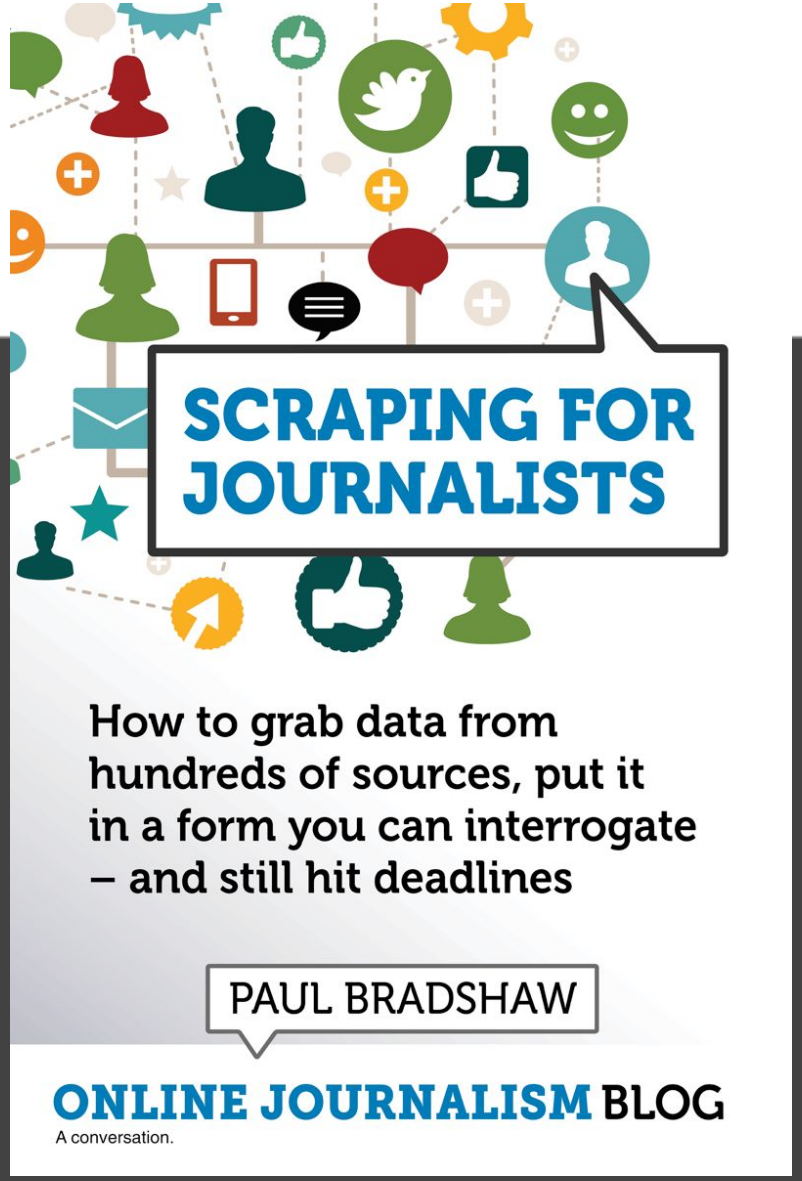
Coding in Google Colab: lists



Paul Bradshaw

Leanpub.com/scrapingforjournalists

What we'll cover

- How to create a **Python notebook** in Google Colab
- Some coding basics: comments, printing, variables, lists and looping
- The difference between strings and numbers and other types of 'object' in Python

1. Go to: **colab.research.google.com**
2. A window will open showing any notebooks
3. Click the **NEW NOTEBOOK** option in the bottom right corner.
4. Give it YOUR NAME!
5. Share in the chat/doc

Variables

- Variables are used in coding to **store information**
- We name a variable when we create it - the name is completely **arbitrary**
- A variable is created (‘***assigned***’) using an equals sign, with the variable name on the left, and the information on the right, like so:
  `mything = “Paul”`
- You can create **different types** of variable - strings, numbers, Booleans, lists, dictionaries - by using certain characters

Different types of (variable) object

Python has a number of data types:

- Strings (words, sentences) - in quotation marks
- Integers (whole numbers)
- Floats (numbers with decimals)
- Booleans: *True* or *False*
- Lists - in square brackets
- Dictionaries - in curly brackets
- Classes. Created especially by a coder: for example a ‘car’ object might have a number of wheels, colour, material etc.

Printing

- **`print()`** can be used to show the results of some calculation or code that would otherwise not be visible to us, e.g. a variable.
- Put what you want to show inside the brackets **`print(numberlist)`**
- ‘Printing’ doesn’t send anything to a printer - it just shows something on the screen
- It can also be used to print text so we know what part of the code is currently running, e.g. **`print(“running scraper now”)`**

Comments

- A **hash (#)** turns code into a comment
- Comments **don't do anything** - they're a way to add notes for yourself and others
- They're normally placed on the line before the code that they're commenting on, e.g.

```
#create a list of numbers
numberlist = [1,2,3]
```

Let’s talk about lists!

Creating a list

- A list is created by using square brackets
- Each item in the list is separated by a comma
- If it's a list of strings, each item needs to be in quotation marks. Lists of numbers don't need quotation marks, however.
- A list is stored like any other variable, by specifying a variable name, followed by the = sign and the list:

```
students = ['Alia', 'George', 'Aida', 'Tom']
```

A list of numbers:

[1, 2, 3, 4, 5]

A list of strings:

[“Birmingham”, “Manchester”, “Liverpool”, “Bristol”, “Glasgow”]

Looping through a list

- To loop through a list you use the **for** command. It looks like this:

```
for student in students:
    This line of code runs for each item in
    the list
```

- The list variable goes between ‘in’ and the colon.

The ‘for loop’ broken down

```
for student in students:
    print(student)
```

- 3 essential parts:
  - **for** followed by a made-up name you use to refer to each item as it loops through
  - **in** followed by the name of the list
  - then the colon **:**
- Any lines you want to run ***while looping*** must be **indented** underneath
- Indented lines will run as many times as there are items in the list

Looping through a list

```
for student in students:
    print(student)
```

- A loop reassigns a variable anew each time (in this example it is called ‘student’)
- The first time, student = “Alia” so it prints “Alia”
- The second time, student = George, so...
- Then ?
- Then ?
- When it reaches the last item it has finished the loop!

A scraping example

```
urls = [“https://www.bbc.co.uk/news/uk-scotland-56072396”,“https://www.bbc.co.uk/news/health-56083905”]
for i in urls:
    scraping_code_goes_here(i)
```

- We store URLs in a list
- A loop goes through that list, and runs the same scraping code on each item (URL)
- It also stores the information it scrapes, each time

Accessing list items: indices

- An index is a position, e.g. 1st, 2nd etc.
- Indices in Python start from 0 and are placed in square brackets *after* the name of a list variable
- So **students[0]** grabs the first item in that list
- And **students[1]** grabs the 2nd item in that list
- You can use negative indexes to grab from the end: **students[-1]** grabs the *last* item in that list
- You can grab a ***range*** of items by specifying the first and last index, separated by a colon: **students[1:3]** - this will also be a list

Recap

- A list is created:

```
students = [‘Alia’, ‘George’, ‘Aida’, ‘Tom’]
```

- Items in that list can be accessed with an index...

```
secondstudent = students[1]
```

- ...or looped through

```
for student in students:
    print(student)
```

Try it now:

- Create a variable to store a list. Remember:
  - Give the variable a name
  - Assign using the = operator
  - Use square brackets to start and end the list
  - Use commas to separate each item
  - Use quotation marks to indicate strings
- Now, see if you can use **print()** to:
  - Print the whole list

Try it now:

- See if you can use a **for loop** to:
  - Print each item

Try it now:

- See if you can use **print()** to:
  - Print just the second item
  - Google how to print the last item!
  - Print the second-to-last item
  - Print the items from 1st to 3rd position

Detour: a bit of coding

- Type this code:

```
myvariable = “blah”
myothervariable = 2
```

- We can’t see those variables yet - we need to print them. Try this:

```
print(myvariable)
```

- How can you print the other variable?
- How can you change a variable?