Output Data as an HTML File with Python (/en/lessons/output-data-as-html-file)

William J. Turkel and Adam Crymble

This lesson takes the frequency pairs created in the 'Counting Frequencies' lesson and outputs them to an HTML file.

Peer-reviewed

edited by

- **Miriam Posner**

reviewed by

- **Jim Clifford**

published

2012-07-17

modified

2012-07-17

difficulty

Medium

This lesson has been translated into Spanish: Salida de datos como archivo HTML con Python (/es/lecciones/salida-de-datos-como-archivo-html)

This lesson was written using Python v. 2.x. Code may not be compatible with newer versions of Python. "Python Introduction and Installation (/lessons/introduction-and-installation)" provides instructions for how you can install Python 2.x alongside newer versions.

This lesson is part of a series. You might want to check out the previous lesson (creating-and-viewing-html-files-with-python).

Contents

Lesson Goals

This lesson takes the frequency pairs created in Counting Frequencies (/lessons/counting-frequencies) and outputs them to an HTML file.

Here you will learn how to output data as an HTML file using Python. You will also learn about string formatting. The final result is an HTML file that shows the keywords found in the original source in order of descending frequency, along with the number of times that each keyword appears.

Files Needed For This Lesson

- `obo.py`

If you do not have these files from the previous lesson, you can download programming-historian-6, a zip file from the previous lesson (/assets/python-lessons6.zip)

Building an HTML wrapper

In the previous lesson, you learned how to embed the message "Hello World!" in HTML tags, write the result to a file and open it automatically in the browser. A program that puts formatting codes around something so that it can be used by another program is sometimes called a *wrapper*. What we're going to do now is develop an HTML wrapper for the output of our code that computes word frequencies. We're also going to add some helpful, dynamic *metadata* to supplement the frequency data collected in Counting Frequencies (/lessons/counting-frequencies).

Metadata

The distinction between data and metadata is crucial to information science. Metadata are data about data. This concept should already be very familiar to you, even if you haven't heard the term before. Consider a traditional book. If we take the text of the book to be the data, there are a number of other characteristics which are associated with that text, but which may or may not be explicitly printed in the book. The title of the work, the author, the publisher, and the place and date of publication are metadata that are typically printed in the work. The place and date of writing, the name of the copy editor, Library of Congress cataloging data, and the name of the font used to typeset the book are sometimes printed in it. The person who purchased a particular copy may or may not write their name in the book. If the book belongs in the collection of a library, that library will keep additional metadata, only some of which will be physically attached to the book. The record of borrowing, for example, is usually kept in some kind of database and linked to the book by a unique identifier. Libraries, archives and museums all have elaborate systems to generate and keep track of metadata.

When you're working with digital data, it is a good idea to incorporate metadata into your own files whenever possible. We will now develop a few basic strategies for making our data files *self-documenting*. In our wrapper, we want to include dynamic information about the file, such as the time and date it was created, as well as an HTML title that is relevant to the file. In this case we could just give it a name ourselves, but when we start working with multiple files, automatically creating self-documenting files will save a lot of time, so we'll practice now. And for that, we'll have to learn to take advantage of a few more powerful string formatting options.

Python string formatting

Python includes a special formatting operator that allows you to insert one string into another one. It is represented by a percent sign followed by an "s". Open a Python shell and try the following examples.

```
frame = 'This fruit is a %s'
print(frame)
-> This fruit is a %s

print(frame % 'banana')
-> This fruit is a banana

print(frame % 'pear')
-> This fruit is a pear
```

There is also a form which allows you to interpolate a list of strings into another one.

```
frame2 = 'These are %s, those are %s'
print(frame2)
-> These are %s, those are %s

print(frame2 % ('bananas', 'pears'))
-> These are bananas, those are pears
```

In these examples, a `%s` in one string indicates that another string is going to be embedded at that point. There are a range of other string formatting codes, most of which allow you to embed numbers in strings in various formats, like `%i` for integer (eg. 1, 2, 3), `%f` for floating-point decimal (eg. 3.023, 4.59, 1.0), and so on. Using this method we can input information that is unique to the file.

Self-documenting data file

Let's bundle some of the code that we've already written into functions. One of these will take a URL and return a string of lowercase text from the web page. Copy this code into the `obo.py` module.

```
# Given a URL, return string of lowercase text from page.

def webPageToText(url):
    import urllib2
    response = urllib2.urlopen(url)
    html = response.read()
    text = stripTags(html).lower()
    return text
```

We're also going to want a function that takes a string of any sort and makes it the body of an HTML file which is opened automatically in Firefox. This function should include some basic metadata, like the time and date that it was created and the name of the program that created it. Study the following code carefully, then copy it into the `obo.py` module.

Mac Instructions

If you are using a Mac, make sure you include the proper file path in the filename variable on the 2nd last line to reflect where you're saving your files.

```
# Given name of calling program, a url and a string to
wrap,
# output string in html body with basic metadata and open
in Firefox tab.

def wrapStringInHTMLMac(program, url, body):
    import datetime
    from webbrowser import open_new_tab

    now = datetime.datetime.today().strftime("%Y%m%d-
%H%M%S")
    filename = program + '.html'
    f = open(filename,'w')

    wrapper = """<html>
    <head>
    <title>%s output - %s</title>
    </head>
    <body><p>URL: <a href=\"%s\">%s</a></p><p>%s</p>
</body>
    </html>"""

    whole = wrapper % (program, now, url, url, body)
    f.write(whole)
    f.close()

    #Change the filepath variable below to match the
location of your directory
    filename =
'file:///Users/username/Desktop/programming-historian/' +
filename

    open_new_tab(filename)
```

Windows Instructions

```
# Given name of calling program, a url and a string to
wrap,
# output string in html body with basic metadata
# and open in Firefox tab.

def wrapStringInHTMLWindows(program, url, body):
    import datetime
    from webbrowser import open_new_tab

    now = datetime.datetime.today().strftime("%Y%m%d-
%H%M%S")

    filename = program + '.html'
    f = open(filename,'w')

    wrapper = """<html>
    <head>
    <title>%s output - %s</title>
    </head>
    <body><p>URL: <a href=\"%s\">%s</a></p><p>%s</p>
</body>
    </html>"""

    whole = wrapper % (program, now, url, url, body)
    f.write(whole)
    f.close()

    open_new_tab(filename)
```
***

Note that this function makes use of the string formatting operator about which we just learned. If you are still having trouble with this idea, take a look at the HTML file that opened in your new Firefox tab and you should see how this worked. If you're still stuck, take a look at the

```
URL: http://www.oldbaileyonline.org/browse.jsp?
id=t17800628-33&div=t17800628-33
```
in the HTML file and trace back how the program knew to put the URL value there.

The function also calls the Python `datetime` library to determine the current time and date. Like the string formatting operator `%s`, this library uses the `%` as replacements for values. In this case, the `%Y %m %d %H %M %S` represents year, month, date, hour, minute and second respectively. Unlike the `%s`, the program will determine the value of these variables for you using your computer's clock. It is important to recognize this difference.

This date metadata, along with the name of the program that called the function, is stored in the HTML title tag. The HTML file that is created has the same name as the Python program that creates it, but with a `.html` extension rather than a `.py` one.

Putting it all together

Now we can create another version of our program to compute frequencies. Instead of sending its output to a text file or an output window, it sends the output to an HTML file which is opened in a new Firefox tab. From there, the program's output can be added easily as bibliographic entries to Zotero. Type or copy the following code into your text editor, save it as `html-to-freq-3.py` and execute it, to confirm that it works as expected.

Use either obo.wrapStringInHTMLMac() or obo.wrapStringInHTMLWindows() as appropriate for your system.

```
# html-to-freq-3.py
import obo

# create sorted dictionary of word-frequency pairs
url = 'http://www.oldbaileyonline.org/browse.jsp?
id=t17800628-33&div=t17800628-33'
text = obo.webPageToText(url)
fullwordlist = obo.stripNonAlphaNum(text)
wordlist = obo.removeStopwords(fullwordlist,
obo.stopwords)
dictionary = obo.wordListToFreqDict(wordlist)
sorteddict = obo.sortFreqDict(dictionary)

# compile dictionary into string and wrap with HTML
outstring = ""
for s in sorteddict:
    outstring += str(s)
    outstring += "<br />"
obo.wrapStringInHTMLMac("html-to-freq-3", url, outstring)
```

Note that we interspersed our word-frequency pairs with the HTML break tag `<br\>`, which acts as a *newline*. If all went well, you should see the same word frequencies that you computed in the last section, this time in your browser window.

Suggested Readings

- Lutz, Learning Python
    - Re-read and review Chs. 1-17

Code Syncing

To follow along with future lessons it is important that you have the right files and programs in your "programming-historian" directory. At the end of each chapter you can download the "programming-historian" zip file to make sure you have the correct code. If you are following along with the Mac / Linux version you may have to open the `obo.py` file and change "file:///Users/username/Desktop/programming-historian/" to the path to the directory on your own computer.

- python-lessons7.zip zip sync (/assets/python-lessons7.zip)

Great! Now you're ready to move on to the next lesson (keywords-in-context-using-n-grams).

About the authors

William J. Turkel is Professor of History at the University of Western Ontario.

Adam Crymble is a senior lecturer of digital history at the University of Hertfordshire.

Suggested Citation

William J. Turkel and Adam Crymble, "Output Data as an HTML File with Python," *The Programming Historian* 1 (2012), https://programminghistorian.org/en/lessons/output-data-as-html-file.

If you wish to print this lesson, we suggest using Chrome for the best appearance.

ISSN 2397-2068 (English) (/)

ISSN 2517-5769 (Spanish) (/es)

ISSN 2631-9462 (French) (/fr)