

## Computing and Software

# Evaluating Scripting Languages: How Python Can Help Political Methodologists

**Holger Döring**

Universität Konstanz

*Holger.Doering@uni-konstanz.de*

## Why Python?

Political methodologists tend to make passionate statements about their software tools. The PolMeth mailing list frequently gives strong advocacy for the use of Linux, L<sup>A</sup>T<sub>E</sub>X, Emacs and other specific programmes. For statistical analysis R has become the mainstream programming language. However, frequent encouragements to use PHP for web purposes or Perl for various scripting tasks highlight the need for a major scripting language beside R. Once political scientists need systematic parsing of markup languages or have to generate web presentations from their data, R quickly reaches its limits. For me, Python has become my favourite scripting language of choice. Having had some previous exposure to C, Java, PHP and Perl, Python turned out to meet all my needs for software development, that R can not fulfil. So let me introduce you to the beauty of Python.

Python helps with almost all of the data management tasks I need. Two applications of the language accompany my every day work: First, I use Python scripts to generate data sets from information provided at internet pages (web scraping). Second, I work with SQLite and Django to manage more complex data sets that require database operations, such as merging, virtual tables, and visualization in web pages. Both of these usages of a modern programming language have increased my productivity significantly and made data resources more easily available. In order to introduce you to Python, I first evaluate contemporary programming languages and their appropriateness for political methodology. Subsequently, I demonstrate how to use Python to generate a data set from an online source. In the last part, I discuss some more advanced issues of data analysis and evaluate how Python can help in a world of ever more easily available online data.

## Evaluating modern programming languages

Python has existed for almost two decades and became popular among programmers in the late nineties<sup>1</sup>. The language is open source software and available at [www.python.org](http://www.python.org). It is preinstalled with Mac OS X and most Linux distributions as well as easily installable on Windows systems. Today, Python is a highly developed, well documented lan-

guage widely used. Much of google is driven by Python. Its development is constantly evolving with a major new version released in October 2008 (Python 3.0). Most important, the language is easy to learn and still satisfies your needs even if you have reached high levels of proficiency. Consequently, it is a good choice for a first programming language to learn.

As with every modern programming language, Python comes with a big standard library that makes many tasks easier, such as reading and parsing files or using regular expressions. In addition, many projects have evolved around Python providing extensions to the language. These packages allow for example rapid development of webpages, advanced numerical analysis or processing of various data types. Most of these additional modules can be easily installed via the Python Package Index ([pypi.python.org/pypi](http://pypi.python.org/pypi)) a repository of software that contains many useful program packages, similar to CRAN. Finally, the language comes with an interpreter, as in R, that allows shell-like exploration of language features.

How does Python perform compared to other software languages? Most of the more traditional programming languages are too complex for our every day tasks in political methodology. Memory management as required in C is something political scientist should not worry about for almost any of their projects. Java may also be too complex and its syntax too verbose for most of our tasks, however it is widely used in agent-based modelling.

PHP and Perl were the most popular scripting languages in the late nineties and the first part of this decade. The syntax of both of these languages can not stand up to the simple beauty of Python code. Perl even has become famous for its idiosyncratic syntax. Its motto of "There's more than one way to do it" has made it known to be the "The Swiss Army Chainsaw" allowing one to hack scripts very quickly. However, Perl scripts are often hard to understand by others or even by oneself after some time has passed. Beside Perl, PHP has been very popular for developing dynamic webpages. However, PHP requires a web server to run and is often less powerful than Perl and Python in not web related tasks. Finally, there are issues of language

<sup>1</sup>To compare the spread of different computer languages see for example the TIOBE Programming Community Index [www.tiobe.com/tpci.htm](http://www.tiobe.com/tpci.htm)

design that make Python a superior choice but I am not the one who can talk about design issues very competently.

Currently, only Ruby has developed to be a major contender of Python's popularity. Especially, its web framework "Ruby on Rails" has generated a significant group of advocates for the language. Ruby and Python are very similar and debates over general benefits of either of these languages quickly turn into very detailed aspects of agile programming languages. Python is in wider use among scientists whereas Ruby is driven significantly by web development projects. NumPy and Python's superb Unicode support, even better in the upcoming Python 3.0, would be my arguments in favour of Python over Ruby. Nevertheless, if you worked with Ruby and feel comfortable with the language, it may not be worthwhile to switch to Python. Either of the two languages fulfils the demands political methodologists have in data processing and web related tasks.

## Data management with Python

Let us now turn to applications of modern scripting languages and demonstrate how they can support our every day computing tasks. More and more data sources for political scientists are available online. Mostly, the information does not come in a neat spreadsheet like format that we can easily import into our statistical packages. Data may be available in a highly structured format such as XML, but most of current online information is provided in HTML. It is up to the data analyst to turn these digital resources into data sets quickly and the task at hand is significantly easier to handle with a modern scripting language.

Recently, Jackman (2006) demonstrated how R can be used to read reasonable structured online sources. However, in my experience, R is quickly stretched to its limits

once the web pages get somewhat more complicated. Solely relying on string processing tools such as regular expressions quickly creates complicated programmes and ignores the structure of the document provided by the markup language. Modern scripting languages come with parsers that make reading structured data easier. These parsers allow very fine grained access to elements of a data source. As reading and analysing structured documents is one of the major tasks for modern scripting languages, these parsers are well developed and different approaches to access the data are implemented.

The structure of most online sources we want to import is very simple. First, we have one or various content pages with links to each page that contains the information we are interested in. Second, data has to be extracted from each of these pages and sometimes the documents contain links to further information we may want to include. As an example, take a website with information on MPs or legislation. While browsing these webpages, we start from navigating index pages to the specific information we are interested in. Hence, our computer programs should do the same to extract the information we need.

Let us start with an example on how to fetch data from nested HTML pages. On the following pages, I provide a Python script, that reads information about all postings on the PolMeth mailing list as provided at <http://polmeth.wustl.edu/polmeth.php> and its subpages (see Figure 1). From this information, we want to generate a data set with information on all postings (author, date, title, URL) at the PolMeth list. In addition, we want to calculate a top ten list of authors posting at PolMeth. I now demonstrate how these nested online data sources can be turned into a data set with a modern scripting language.

LIST BY MONTH	
MAY-2003	
	2008: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a>
	2007: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a> <a href="#">Apr</a> <a href="#">May</a> <a href="#">Jun</a> <a href="#">Jul</a> <a href="#">Aug</a> <a href="#">Sep</a> <a href="#">Oct</a> <a href="#">Nov</a> <a href="#">Dec</a>
	2006: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a> <a href="#">Apr</a> <a href="#">May</a> <a href="#">Jun</a> <a href="#">Jul</a> <a href="#">Aug</a> <a href="#">Sep</a> <a href="#">Oct</a> <a href="#">Nov</a> <a href="#">Dec</a>
<b>Date Posted</b>	<b>From</b>
2003-05-13 16:56:25	"Mister
2003-05-13 16:57:33	Jeff Gill
2003-05-13 16:58:52	George
2003-05-13 17:00:30	Gary Kl
2003-05-14 09:58:28	George
2003-05-15 12:27:07	"Cindy
2003-05-20 11:51:50	Suzanne Mueller <smueller@vera.org>
2003-05-20 11:53:22	Suzanne Mueller <smueller@vera.org>
2003-05-20 11:55:38	"Paul F. Manna" <pmanna@polisci.wisc.edu>
2003-05-21 09:57:50	"Franzese, Robert" <franzese@umich.edu>
2003-05-23 08:46:13	"Sharron Lawrence" <Sharron.Lawrence@tandf.co.uk>
2003-05-27 19:35:06	James Gimpel <JGIMPEL@GVPT.UMD.EDU>
2003-05-28 13:27:26	"Franzese, Robert" <franzese@umich.edu>
	2005: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a> <a href="#">Apr</a> <a href="#">May</a> <a href="#">Jun</a> <a href="#">Jul</a> <a href="#">Aug</a> <a href="#">Sep</a> <a href="#">Oct</a> <a href="#">Nov</a> <a href="#">Dec</a>
	2004: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a> <a href="#">Apr</a> <a href="#">May</a> <a href="#">Jun</a> <a href="#">Jul</a> <a href="#">Aug</a> <a href="#">Sep</a> <a href="#">Oct</a> <a href="#">Nov</a> <a href="#">Dec</a>
	2003: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a> <a href="#">Apr</a> <a href="#">May</a> <a href="#">Jun</a> <a href="#">Jul</a> <a href="#">Aug</a> <a href="#">Sep</a> <a href="#">Oct</a> <a href="#">Nov</a> <a href="#">Dec</a>
	2002: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a> <a href="#">Apr</a> <a href="#">May</a> <a href="#">Jun</a> <a href="#">Jul</a> <a href="#">Aug</a> <a href="#">Sep</a> <a href="#">Oct</a> <a href="#">Nov</a> <a href="#">Dec</a>
	2001: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a> <a href="#">Apr</a> <a href="#">May</a> <a href="#">Jun</a> <a href="#">Jul</a> <a href="#">Aug</a> <a href="#">Sep</a> <a href="#">Oct</a> <a href="#">Nov</a> <a href="#">Dec</a>
	2000: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a> <a href="#">Apr</a> <a href="#">May</a> <a href="#">Jun</a> <a href="#">Jul</a> <a href="#">Aug</a> <a href="#">Sep</a> <a href="#">Oct</a> <a href="#">Nov</a> <a href="#">Dec</a>
	1999: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a> <a href="#">Apr</a> <a href="#">May</a> <a href="#">Jun</a> <a href="#">Jul</a> <a href="#">Aug</a> <a href="#">Sep</a> <a href="#">Oct</a> <a href="#">Nov</a> <a href="#">Dec</a>
	1998: <a href="#">Jan</a> <a href="#">Feb</a> <a href="#">Mar</a> <a href="#">Apr</a> <a href="#">May</a> <a href="#">Jun</a> <a href="#">Jul</a> <a href="#">Aug</a> <a href="#">Sep</a> <a href="#">Oct</a> <a href="#">Nov</a> <a href="#">Dec</a>
	<a href="#">A Logocentric Analysis of Our New Logo</a>
	<a href="#">Paper Posting</a>
	<a href="#">Statistics and Voting</a>
	<a href="#">3-D graphical display</a>
	<a href="#">RE: Speical Elections Issue of Statistical Science</a>
	<a href="#">Dirksen Congressional Center Communicator Update</a>
	<a href="#">H-POLMETH: Postdoctoral Fellowship on Race, Crime and Justice</a>
	<a href="#">H-POLMETH: Postdoctoral Fellowship on Race, Crime, and Justice - CORRECTION</a>
	<a href="#">H-POLMETH: NGA Newsletter</a>
	<a href="#">Re: SARA / Journal of Political Ideologies</a>
	<a href="#">H-POLMETH: SARA - Scholarly Articles Research Alerting</a>
	<a href="#">H-POLMETH: Dueling political scientists</a>
	<a href="#">Re: FW: Dueling political scientists (fwd)</a>

Figure 1: PolMeth mailing list archive

To fetch all postings from the PolMeth mailing list, I wrote a Python program that parses data from the online source. For this task, we rely on functions provided in the Python standard library. The library comes with an HTML parser of its own that is a little clumsy. An additional Python package that parses HTML documents, BeautifulSoup ([www.crummy.com/software/BeautifulSoup](http://www.crummy.com/software/BeautifulSoup)), is easier to use and shows better performance with invalid HTML pages. In my work, I have had very good experiences using BeautifulSoup to parse very different online sources.

The following script fetches the data from the PolMeth list web page we are interested in. It proceeds by first reading the index page <http://polmeth.wustl.edu/polmeth.php> and extracts all links to the monthly summaries—links starting with `/mailinglist/search.php`. Subsequently, the program loops over all links, reads the related pages and extracts the information we are interested in. To read the information about a posting, the program processes every table row and extracts information from column entries and hyperlinks provided for posting. Finally, results are stored in a list structure.

#### *Web scraping: An example*

```
#/usr/bin/env python
import codecs, csv, re, urllib
from BeautifulSoup import BeautifulSoup

polmethurl = 'http://polmeth.wustl.edu'

## read online file and parse html
s = urllib.urlopen(polmethurl + '/polmeth.php')
index = BeautifulSoup(s.read())

regex = re.compile('~mailinglist/search.php')
urlmonths = index.findAll(attrs={'href': regex})

## process monthly section of postings
data = []; m = {} # initialize elements
for url in urlmonths:

## read page for current month
    url = polmethurl + url['href']
    print 'fetching ' + url
    s = urllib.urlopen(url)
    mails = BeautifulSoup(s.read())

## process table rows (skip headline)
    for mail in mails.table('tr')[1:]:
        entries = mail('td') # row elements into list

## extract information from row elements
        m['date'], m['time'] = entries[0].string.split()
        m['title'] = entries[2].a.string.strip()

        m['url'] = entries[2].a['href']
        m['url'] = polmethurl + '/mailinglist/' + m['url']

## extract author name from first field (if existing)
        m['authorfull'] = entries[1].string.strip()
        if m['authorfull'].find('&lt;') != -1:
            m['author'] = m['authorfull'].split('&lt;')[0].strip(' ')
        else:
            m['author'] = m['authorfull']

## add elements to list of data
        data.append(m.copy())
```

```
else:
    m['author'] = m['authorfull']

## add elements to list of data
data.append(m.copy())
```

Looking at the program listings you may have quickly realised that Python uses indentions to separate program blocks. This helps to write readable code. In the program, we use lists and dictionaries to assign our data. At the end of the program, information we are interested in is saved in a list where each entry contains information about one posting at the PolMeth list. Every entry has an element for the following information: author, date, time, authorfull, title, url.

After we have processed the data from our online sources, we have to decide if we want to perform our data analysis in Python as well. In the Python community, there is the NumPy package, that allows mathematical array processing and is widely used by scientists. For our current task, determining the top ten posting authors, python standard tools are sufficient.

```
## determine the number of postings per author and sort results
authorlist = [x['author'] for x in data]
authors = [(authorlist.count(x), x) for x in set(authorlist)]
authors.sort()
authors.reverse()

print '\nTop ten postings on PolMeth list'
print '-----'
for i, j in authors[0:9]:
    print str(i) + '\t' + j
```

The listings show that processing spreadsheet like data in python is rather awkward. Here, we rely on list comprehension to generate a variable with the information we are interested in, the number of postings per author.

For me, once I have downloaded and parsed my data from online sources I quickly change horses and turn to R for data analysis. Python is very powerful to process online sources and text files. However, once I have converted these information into something that is more spreadsheet like, R feels more natural.

In order to export our data from the Python program we generate a csv file. Python's syntax for exporting data into a csv file is somewhat verbose. For completeness, I give the listing here:

```
## write all data into csv file
outfile = codecs.open("polmeth.csv", "wb", "utf-8")

cols = ['author', 'date', 'time', 'authorfull', 'title', 'url']
writer = csv.DictWriter(outfile, cols, quoting=csv.QUOTE_NONNUMERIC)
writer.writerow(dict(zip(cols, cols)))
for i in data:
    try:
        writer.writerow(i)
```

```
except:
    "Print can't write row" + i['date'] + i['time']

outfile.close()
```

You can also write the information directly into a database, such as MySQL or SQLite. Once you have exported the data from Python you can proceed by analysing the information in your favourite statistical package. For our example, generating the top ten list of authors posting at PolMeth is straight forward in R. It just requires three lines of code:

```
pm <- read.csv("polmeth.csv", as.is=TRUE)
authors <- sort(tapply(pm$author, pm$author, length),
               decreasing = TRUE)
authors[1:10]
```

Maybe, you have gained some ideas how Python, or any other modern scripting language, may help to derive data from structured information sources. In the previous examples, we used Python to extract data from the PolMeth list archive. Relying on a HTML parser that comes with any high level language allows fine grained access to page elements. Running the scripts, you may have figured out what the top ten list looks like.

Future online presentations will provide information that is even better structured (eg. XML, JSON, etc.), hence easier to access. Hopefully, I have convinced you that modern scripting languages are extremely helpful to turn various data sources into data sets. My discussion of recent trends in data provision at the last section will explore more applications for modern scripting languages. Before, let me shortly present some more tasks I manage with Python.

#### *Organising and presenting data*

I regularly use Python to convert online data into data sets. In addition, I apply the language to process textual data and for record linkage of different data sets. The latter requires use of a simple fuzzy string matching algorithms. The language also helps with a couple of minor tasks that show up every now and then, such as renaming multiple files, data conversion etc. For most of these requirements I can rely solely on Python's standard library and sometimes I add external open source packages. I rarely have to code extensively to fulfil my data management tasks.

Some of my data sets have become rather complex and require a database design to be managed coherently. I have started to use the Python web framework Django ([www.djangoproject.com](http://www.djangoproject.com)) in order to develop a web interface for this data. Web application frameworks allow one to create dynamic websites easily. Once you want to provide an interface to more complex data structures, frameworks allow you to develop a web interface quickly. Web frameworks help with accessing your data from a database with user management and templating of HTML pages. Django, a popular Python framework, has allowed me to manage my

data very easily. In my view, it is easier and faster to develop a web presentation in Django than in PHP, which is often used for these tasks. You will get familiar with Django quickly if you have some knowledge of Python.

Building web applications for data sets is not a major task for political methodologists. However, once you work with different people on the creation of a data set, web based data coding makes the data generation more reliable. Web frameworks also allow you to include modern features of webpages, such as wikis and comment sections. In addition, see the shift from paper based to Internet based uses of surveys. Nowadays, everyone can setup a small online survey with very limited resources. As this article promotes the usage of Python for data related tasks, web frameworks that are provided for modern scripting languages make the creation of data presentations online significantly easier.

### **Trends in data provision and analysis**

There are current trends in the online world that will make knowledge of a modern scripting language even more important in the coming years. There are web pages that provide systematic data on questions of political science and these data sources are regularly updated. Combining these data sources allows one to generate data sets for studies of political science automatically.

Take for example the web page at [projects.washingtonpost.com/congress](http://projects.washingtonpost.com/congress), a so called mashup. The page collects and presents information on voting in the Congress from various online sources. It is important to note that the providers do not code any of the data themselves. They just combine existing online sources. The page has been created with Python and Django, tools I have previously presented. Political scientists that want to make systematic use of data in a similar way need programmed scripts that include these data sources automatically or download them at regular intervals. Modern data generation does not require one to code extensive amounts of data manually but to code a computer script that extracts information. Hence, political methodologists have to gain knowledge of a scripting language in order to write these scripts.

There is a strong trend in the online community towards a more systematic distinction between data and layout. This trend is referred to as the semantic web or web 3.0. Out of this work data will be presented in a way that allows a combination of different data sources. One interesting project in this respect is [www.freebase.com](http://www.freebase.com). Whereas Wikipedia provides a huge data source of information online, its different information is difficult to extract automatically. Freebase provides an open database that can be extended by its users and the project provides an open interface to systematically extract relevant information. In the same line, Wikipedia has also started to include some

semantic information in its articles.

Other online sources are also provided in a more systematic way. I do not want to go into too much detail. Creating data sets by drawing on systematically organised online sources will be a substantial part of our future work on data generation. Contrary, to the approach that I provided in my example, these data sources can be read by systematically specifying the content of the element to be extracted instead of relying on format parameters such as the table entry as used in my example.

Combining and analysing the ever growing amount of information available has led to new methods of data mining. Segaran (2007) gives a nice and accessible demonstration of how Python can be used to analyse different online information. He provides examples of modern data mining techniques applied to various online information that provide systematic interfaces to their data (APIs). Segaran's book shows how you access the online resources via Python and discusses different data mining algorithms to analyse these data. The scripts are short and easy to read, most of the statistical techniques are similar to the ones we apply in political methodology. The book gives many inspirations on how to make use of new opportunities provided through structured online data.

To make systematic use of modern methods of data provision and analysis you need to have some knowledge of a powerful scripting language. All these languages come with package repositories that provide many scripts to work with online data and to access various web resources. To turn the information you are interested into a data set requires you to include these packages in your own script and to modify them for your needs.

## Conclusion

Hopefully, this note was sufficient to convince you of the benefits that modern scripting languages provide for political methodologists. In my opinion, Python and Ruby have the right balance between power and complexity for all programming tasks that statistical programming languages can not fulfil. Both languages are easy to learn and are still powerful programming languages once you have gained more proficiency. It may well be that every decade has its programming language and I believe that Python and Ruby are today's languages for computing tasks in political methodology. Knowing one of these modern scripting languages becomes even more important as online data sources become more numerous and better structured. A powerful scripting language at hand will help you to draw on this information quickly and adds to many more of your scripting needs. If I have convinced you on the power of Python, pick up Chun (2007) to get a thorough introduction into the language and its applications.

## References

- Chun, Wesley J. 2007. *Core python programming*. Upper Saddle River, NJ: Prentice Hall.
- Jackman, Simon. 2006. "Data from the web into R." *The Political Methodologist* 14(2): 11-15.
- Segaran, Toby. 2007. *Programming collective intelligence: Building smart Web 2.0 applications*. Sebastopol, CA: O'Reilly Media.

## Professional Development

### Introducing the Advice to Junior Faculty Column: Advice from Gary King

**Corrine McConnaughey**

The Ohio State University

*mcconnaughey.3@polisci.osu.edu*

Junior faculty members of the Society for Political Methodology can find themselves in real need of mentorship and advice from senior colleagues perhaps even more so than graduate students. Some junior faculty members are the only political methodologists in their home departments. Others are not so alone, but do not always feel safe to ask questions about their career considerations of their own senior colleagues; doing so in their first year or two on the job, when they have yet to forge relationships with those colleagues,

may be especially daunting. In response to these concerns, SPM Long Range Planning Committee proposed an advice column in *The Political Methodologist* as one venue for increased mentorship of junior faculty by senior SPM members.

To ensure the relevance of the column to our junior faculty members, I contacted a diverse set of junior faculty SPM members to solicit suggested questions or topics. I