Python and the Natural Language Toolkit

Why Python?

Python is a simple yet powerful programming language with excellent functionality for processing linguistic data. Python can be downloaded for free from `http://www.python.org/`.

Here is a five-line Python program which takes text input and prints all the words ending in `ing`:

```
>>> import sys                          # load the system library
>>> for line in sys.stdin:              # for each line of input
...     for word in line.split():       # for each word in the line
...         if word.endswith('ing'):    # does the word end in 'ing'?
...             print word              # if so, print the word
```

This program illustrates some of the main features of Python. First, whitespace is used to *nest* lines of code, thus the line starting with `if` falls inside the scope of the previous line starting with `for`, so the `ing` test is performed for each word. Second, Python is *object-oriented*; each variable is an entity which has certain defined attributes and methods. For example, `line` is more than a sequence of characters. It is a string object that has a **method** (or operation) called `split` that we can use to break a line into its words. To apply a method to an object, we give the object name, followed by a period, followed by the method name. Third, methods have *arguments* expressed inside parentheses. For instance, `split` had no argument because we were splitting the string wherever there was white space. To split a string into sentences delimited by a period, we could write `split('.')`. Finally, and most importantly, Python is highly readable, so much so that it is fairly easy to guess what the above program does even if you have never written a program before.

We chose Python as the implementation language for NLTK because it has a shallow learning curve, its syntax and semantics are transparent, and it has good string-handling functionality. As a scripting language, Python facilitates interactive exploration. As an object-oriented language, Python permits data and methods to be encapsulated and re-used easily. As a dynamic language, Python permits attributes to be added to objects on the fly, and permits variables to be typed dynamically, facilitating rapid development. Python comes with an extensive standard library, including components for graphical programming, numerical processing, and web data processing.

Python is heavily used in industry, scientific research, and education around the world. Python is often praised for the way it facilitates productivity, quality, and maintainability of software. A collection of Python success stories is posted at `http://www.python.org/about/success/`.

NLTK defines a basic infrastructure that can be used to build NLP programs in Python. It provides: basic classes for representing data relevant to natural language processing; standard interfaces for performing tasks such as tokenization, tagging, and parsing; standard implementations for each task which can be combined to solve complex problems; and extensive documentation including tutorials and reference documentation.

Teaching and Learning Python and NLTK

This book contains self-paced learning materials including many examples and exercises. An effective way for students to learn is simply to work through the materials, with the help of other students and instructors. The program fragments can be cut and pasted directly from the online tutorials. The HTML version has a blue bar beside each program fragment; click on the bar to automatically copy the program fragment to the clipboard (assumes appropriate browser security settings.) Note that the examples in each chapter often use functions and modules that were imported in earlier examples. When lifting examples from these materials it is often necessary to add one or more `import` statements.

This material can also be used as the basis for lecture-style presentations (and some slides are available for download from the NLTK website). We believe that an effective way to present the materials is through interactive presentation of the examples, entering them at the Python prompt, observing what they do, and modifying them to explore some empirical or theoretical question.

Integrated Development Environments: The easiest way to develop Python code, and to perform interactive Python demonstrations, is to use the simple editor and interpreter GUI that comes with Python called *IDLE*, the *Integrated DeveLopment Environment for Python*. A more sophisticated approach is to use a full-blown IDE such as *Eclipse* together with a Python plugin such as *PyDEV* (see the NLTK wiki for instructions).

NLTK Community: NLTK has a large and growing user base. There are mailing lists for annoucements about NLTK, for developers and for teachers. A wiki hosted on the NLTK website is available where registered users can share 'recipes', convenient short scripts for performing useful tasks. The wiki also lists courses around the world where NLTK has been adopted, serving as a useful source of associated resources.

The Design of NLTK

NLTK was designed with six requirements in mind:

Simplicity: We have tried to provide an intuitive and appealing framework along with substantial building blocks, for students to gain a practical knowledge of NLP without getting bogged down in the tedious house-keeping usually associated with processing annotated language data. We have provided software distributions for several platforms, along with platform-specific instructions, to make the toolkit easy to install.

Consistency: We have made a significant effort to ensure that all the data structures and interfaces are consistent, making it easy to carry out a variety of tasks using a uniform framework.

Extensibility: The toolkit easily accommodates new components, whether those components replicate or extend existing functionality. Moreover, the toolkit is organized so that it is usually obvious where extensions would fit into the toolkit's infrastructure.

Modularity: The interaction between different components of the toolkit uses simple, well-defined interfaces. It is possible to complete individual projects using small parts of the toolkit, without needing to understand how they interact with the rest of the toolkit. This allows students to learn how to use the toolkit incrementally throughout a course. Modularity also makes it easier to change and extend the toolkit.

Well-Documented: The toolkit comes with substantial documentation, including nomenclature, data structures, and implementations.

Contrasting with these requirements are three non-requirements — potentially useful features that we have deliberately avoided. First, while the toolkit provides a wide range of functions, it is not intended to be encyclopedic. There should be a wide variety of ways in which students can extend the toolkit. Second, while the toolkit should be efficient enough that students can use their NLP systems to perform meaningful tasks, it does not need to be highly optimized for runtime performance. Such optimizations often involve more complex algorithms, and sometimes require the use of C or C++. This would make the toolkit less accessible and more difficult to install. Third, we have avoided clever programming tricks, since clear implementations are far preferable to ingenious yet indecipherable ones.

NLTK Organization: NLTK is organized into a collection of task-specific packages. Each package is a combination of data structures for representing a particular kind of information such as trees, and implementations of standard algorithms involving those structures such as parsers. This approach is a standard feature of *object-oriented design*, in which components encapsulate both the resources and methods needed to accomplish a particular task.

The most fundamental NLTK components are for identifying and manipulating individual words of text. These include: `tokenize`, for breaking up strings of characters into word tokens; `tag`, for adding part-of-speech tags, including regular-expression taggers, n-gram taggers and Brill taggers; and the Porter stemmer.

The second kind of module is for creating and manipulating structured linguistic information. These components include: `tree`, for representing and processing parse trees; `featurestructure`, for building and unifying nested feature structures (or attribute-value matrices); `cfg`, for specifying context-free grammars; and `parse`, for creating parse trees over input text, including chart parsers, chunk parsers and probabilistic parsers.

Several utility components are provided to facilitate processing and visualization. These include: `draw`, to visualize NLP structures and processes; `probability`, to count and collate events, and perform statistical estimation; and `corpora`, to access tagged linguistic corpora.

A further group of components is not part of NLTK proper. These are a wide selection of third-party contributions, often developed as student projects at various institutions where NLTK is used, and distributed in a separate package called *NLTK Contrib*. Several of these student contributions, such as the Brill tagger and the HMM module, have now been incorporated into NLTK. Although these contributed components are not maintained, they may serve as a useful starting point for future student projects.

In addition to software and documentation, NLTK provides substantial corpus samples. Many of these can be accessed using the `corpora` module, avoiding the need to write specialized file parsing code before you can do NLP tasks. These corpora include: Brown Corpus — 1.15 million words of tagged text in 15 genres; a 10% sample of the Penn Treebank corpus, consisting of 40,000 words of syntactically parsed text; a selection of books from Project Gutenberg totally 1.7 million words; and other corpora for chunking, prepositional phrase attachment, word-sense disambiguation, information extraction

Note on NLTK-Lite: Since mid-2005, the NLTK developers have been creating a lightweight version NLTK, called NLTK-Lite. NLTK-Lite is simpler and faster than NLTK. Once it is complete, NLTK-Lite will provide the same functionality as NLTK. However, unlike NLTK, NLTK-Lite does not impose such a heavy burden on the programmer. Wherever possible, standard Python objects are used instead of custom NLP versions, so that students learning to program for the first time will be learning to program in Python with some useful libraries, rather than learning to program in NLTK.

NLTK Papers: NLTK has been presented at several international conferences with published proceedings, as listed below:

Edward Loper and Steven Bird (2002). NLTK: The Natural Language Toolkit, *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, Somerset, NJ: Association for Computational Linguistics, pp. 62-69, http://arXiv.org/abs/cs/0205028

Steven Bird and Edward Loper (2004). NLTK: The Natural Language Toolkit, *Proceedings of the ACL demonstration session*, pp 214-217. http://eprints.unimelb.edu.au/archive/00001448/

Steven Bird (2005). NLTK-Lite: Efficient Scripting for Natural Language Processing, *4th International Conference on Natural Language Processing*, pp 1-8. http://eprints.unimelb.edu.au/archive/00001453/

Steven Bird (2006). NLTK: The Natural Language Toolkit, *Proceedings of the ACL demonstration session* http://www.ldc.upenn.edu/sb/home/papers/nltk-demo-06.pdf

Edward Loper (2004). NLTK: Building a Pedagogical Toolkit in Python, *PyCon DC 2004* Python Software Foundation, http://www.python.org/pycon/dc2004/papers/

Ewan Klein (2006). Computational Semantics in the Natural Language Toolkit, *Australian Language Technology Workshop*. http://www.alta.asn.au/events/altw2006/proceedings/Klein.pdf