Preface

This is a book about **Natural Language Processing**. By **natural language** we mean a language that is used for everyday communication by humans; languages like English, Hindi or Portuguese. In contrast to artificial languages such as programming languages and mathematical notations, natural languages have evolved as they pass from generation to generation, and are hard to pin down with explicit rules. We will take Natural Language Processing (or NLP for short) in a wide sense to cover any kind of computer manipulation of natural language. At one extreme, it could be as simple as counting the number of times the letter *t* occurs in a paragraph of text. At the other extreme, NLP involves "understanding" complete human utterances, at least to the extent of being able to give useful responses to them.

Technologies based on NLP are becoming increasingly widespread. For example, handheld computers (PDAs) support predictive text and handwriting recognition; web search engines give access to information locked up in unstructured text; machine translation allows us to retrieve texts written in Chinese and read them in Spanish. By providing more natural human-machine interfaces, and more sophisticated access to stored information, language processing has come to play a central role in the multilingual information society.

This book provides a comprehensive introduction to the field of NLP. It can be used for individual study or as the textbook a course on natural language processing or computational linguistics. The book is intensely practical, containing hundreds of fully-worked examples and graded exercises. It is based on the Python programming language together with an open source library called the *Natural Language Toolkit* (NLTK). NLTK includes software, data, and documentation, all freely downloadable from http://nltk.org/. Distributions are provided for Windows, Macintosh and Unix platforms. We encourage you, the reader, to download Python and NLTK, and try out the examples and exercises along the way.

Audience

NLP is important for scientific, economic, social, and cultural reasons. NLP is experiencing rapid growth as its theories and methods are deployed in a variety of new language technologies. For this reason it is important for a wide range of people to have a working knowledge of NLP. Within industry, it includes people in *human-computer interaction*, *business information analysis*, and *Web software development*. Within academia, this includes people in areas from *humanities computing* and *corpus linguistics* through to *computer science* and *artificial intelligence*.

This book is intended for a diverse range of people who want to learn how to write programs that analyze written language:

New to Programming?: The book is suitable for readers with no prior knowledge of programming, and the early chapters contain many examples that you can simply copy and try for yourself, together with graded exercises. If you decide you need a more general introduction to Python, we recommend you read *Learning Python* (O'Reilly) in conjunction with this book.

New to Python?: Experienced programmers can quickly learn enough Python using this book to get immersed in natural language processing. All relevant Python features are carefully explained and exemplified, and you will quickly come to appreciate Python's suitability for this application area.

Already dreaming in Python?: Skip the Python examples and dig into the interesting language analysis material that starts in Chapter 1. Soon you'll be applying your skills to this exciting new application area.

Emphasis

This book is a **practical** introduction to NLP. You will learn by example, write real programs, and grasp the value of being able to test an idea through implementation. If you haven't learnt already, this book will teach you **programming**. Unlike other programming books, we provide extensive illustrations and exercises from NLP. The approach we have taken is also **principled**, in that we cover the theoretical underpinnings and don't shy away from careful linguistic and computational analysis. We have tried to be **pragmatic** in striking a balance between theory and application, identifying the connections and the tensions. Finally, we recognize that you won't get through this unless it is also **pleasurable**, so we have tried to include many applications and examples that are interesting and entertaining, sometimes whimsical.

What You Will Learn

By digging into the material presented here, you will learn:

- how simple programs can help you manipulate and analyze language data, and how to write these programs;
- how key concepts from NLP and linguistics are used to describe and analyse language;
- how data structures and algorithms are used in NLP;
- how language data is stored in standard formats, and how data can be used to evaluate the performance of NLP techniques.

Depending on your background, and your motivation for being interested in NLP, you will gain different kinds of skills and knowledge from this book, as set out below:

Goals	Background in Arts and Humanities	Background in Science and Engineering
Language	Programming to manage language	Language as a source of interesting
Analysis	data, explore linguistic models, models, and test empirical claims	problems in data modeling, data mining, and knowledge discovery
Language	Learning to program, with	Knowledge of linguistic algorithms
Technology	applications to familiar problems to work in language technology or other technical field	and data structures for high quality, maintainable language processing software

Table 1:

Organization

The early chapters are organized in order of conceptual difficulty, starting with a gentle introduction to language processing and Python, before proceeding on to fundamental topics such as tokenization, tagging, and evaluation. After this, a sequence of chapters covers topics in grammars and parsing, which have long been central tasks in language processing. The last third of the book contains chapters on advanced topics, which can be read independently of each other.

Each chapter consists of an introduction, a sequence of sections that progress from elementary to advanced material, and finally a summary and suggestions for further reading. Most sections include exercises that are graded according to the following scheme: ☼ is for easy exercises that involve minor modifications to supplied code samples or other simple activities; ◑ is for intermediate exercises that explore an aspect of the material in more depth, requiring careful analysis and design; ★ is for difficult, open-ended tasks that will challenge your understanding of the material and force you to think independently (readers new to programming are encouraged to skip these); ☺ is for non-programming exercises for reflection or discussion. The exercises are important for consolidating the material in each section, and we strongly encourage you to try a few before continuing with the rest of the chapter.

Within each chapter, we'll be switching between different styles of presentation. In one style, natural language will be the driver. We'll analyze language, explore linguistic concepts, and use programming examples to support the discussion. Sometimes we'll present Python constructs that have not been introduced systematically; this way you will see useful idioms early, and might not appreciate their workings until later. In the other style, the programming language will be the driver. We'll analyze programs, explore algorithms, and use linguistic examples to support the discussion.

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/. Installers are available for all platforms.

Here is a five-line Python program that processes `file.txt` and prints all the words ending in `ing`:

```
>>> for line in open("file.txt"):       # for each line of input text
...     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`; this ensures that the `ing` test is performed for each word. Second, Python is *object-oriented*; each variable is an entity that has certain defined attributes and methods. For example, the value of the variable `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 write the object name, followed by a period, followed by the method name; i.e., `line.split()`. Third, methods have *arguments* expressed inside parentheses. For instance, in the example above, `split()` had no argument because we were splitting the string wherever there was white space, and we could therefore use empty parentheses. To split a string into sentences delimited by a period, we would 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 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 an 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, part-of-speech tagging, and syntactic parsing; and standard implementations for each task which can be combined to solve complex problems.

NLTK comes with extensive documentation. In addition to this book, the website http://nltk.org/ provides API documentation which covers every module, class and function in the toolkit, specifying parameters and giving examples of usage. The website also provides module **guides**; these contain extensive examples and test cases, and are intended for users, developers and instructors.

Learning Python for Natural Language Processing

This book contains self-paced learning materials including many examples and exercises. An effective way to learn is simply to work through the materials. The program fragments can be copied directly into a Python interactive session. Any questions concerning the book, or Python and NLP more generally, can be posted to the NLTK-Users mailing list (see http://nltk.org/).

Python Environments: The easiest way to start developing Python code, and to run interactive Python demonstrations, is to use the simple editor and interpreter GUI that comes with Python called *IDLE*, the *Integrated DeveLopment Environment for Python*.

NLTK Community: NLTK has a large and growing user base. There are mailing lists for announcements about NLTK, for developers and for teachers. http://nltk.org/ lists many courses around the world where NLTK and materials from this book have been adopted, a useful source of extra materials including slides and exercises.

The Design of NLTK

NLTK was designed with four primary goals in mind:

Simplicity: We have tried to provide an intuitive and appealing framework along with substantial building blocks, so you can 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. Modularity also makes it easier to change and extend the toolkit.

Contrasting with these goals 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 encyclopedic; it will continue to evolve with the field of NLP. Second, while the toolkit should be efficient enough to support 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 programming languages like C or C++. This would make the toolkit less accessible and more difficult to install. Third, we have tried to avoid clever programming tricks, since clear implementations are preferable to ingenious yet indecipherable ones.

For Instructors

Natural Language Processing (NLP) is often taught within the confines of a single-semester course at advanced undergraduate level or postgraduate level. Many instructors have found that it is difficult to cover both the theoretical and practical sides of the subject in such a short span of time. Some courses focus on theory to the exclusion of practical exercises, and deprive students of the challenge and excitement of writing programs to automatically process language. Other courses are simply designed to teach programming for linguists, and do not manage to cover any significant NLP content. NLTK was originally developed to address this problem, making it feasible to cover a substantial amount

of theory and practice within a single-semester course, even if students have no prior programming experience.

A significant fraction of any NLP syllabus deals with algorithms and data structures. On their own these can be rather dry, but NLTK brings them to life with the help of interactive graphical user interfaces making it possible to view algorithms step-by-step. Most NLTK components include a demonstration which performs an interesting task without requiring any special input from the user. An effective way to deliver the materials is through interactive presentation of the examples, entering them in a Python session, observing what they do, and modifying them to explore some empirical or theoretical issue.

The book contains hundreds of examples and exercises which can be used as the basis for student assignments. The simplest exercises involve modifying a supplied program fragment in a specified way in order to answer a concrete question. At the other end of the spectrum, NLTK provides a flexible framework for graduate-level research projects, with standard implementations of all the basic data structures and algorithms, interfaces to dozens of widely used data-sets (corpora), and a flexible and extensible architecture. Additional support for teaching using NLTK is available on the NLTK website, and on a closed mailing list for instructors.

We believe this book is unique in providing a comprehensive framework for students to learn about NLP in the context of learning to program. What sets these materials apart is the tight coupling of the chapters and exercises with NLTK, giving students — even those with no prior programming experience — a practical introduction to NLP. After completing these materials, students will be ready to attempt one of the more advanced textbooks, such as *Speech and Language Processing*, by Jurafsky and Martin (Prentice Hall, 2008).

This book presents programming concepts in an unusual order, beginning with a non-trivial data type — lists of strings — before introducing non-trivial control structures like comprehensions and conditionals. These idioms permit us to do useful language processing from the start. Once this motivation is in place we deal with the fundamental concepts systematically. Thus we cover the same ground as more conventional approaches, without expecting readers to be interested in the programming language for its own sake.

Chapter	**Arts and Humanities**	**Science and Engineering**
1 Language Processing and Python	2-4	2
2 Text Corpora and Lexical Resources	2-4	2
3 Processing Raw Text	2-4	2
4 Categorizing and Tagging Words	2-4	2
5 Data-Intensive Language Processing	0-2	2
6 Structured Programming	2-4	1
7 Partial Parsing and Interpretation	2	2
8 Grammars and Parsing	3-6	2-4
9 Advanced Parsing	1-4	3
10-14 Advanced Topics	2-4	2-16
Total	18-36	18-36

Table 2: Suggested Course Plans; Lectures/Lab Sessions per Chapter

Acknowledgments

NLTK was originally created as part of a computational linguistics course in the Department of Computer and Information Science at the University of Pennsylvania in 2001. Since then it has been developed and expanded with the help of dozens of contributors. It has now been adopted in courses in dozens of universities, and serves as the basis of many research projects.

In particular, we're grateful to the following people for their feedback, comments on earlier drafts, advice, contributions: Michaela Atterer, Greg Aumann, Kenneth Beesley, Steven Bethard, Ondrej Bojar, Trevor Cohn, Grev Corbett, James Curran, Jean Mark Gawron, Baden Hughes, Gwillim Law, Mark Liberman, Christopher Maloof, Stefan Müller, Stuart Robinson, Jussi Salmela, Rob Speer. Many others have contributed to the toolkit, and they are listed at http://nltk.org/. We are grateful to many colleagues and students for feedback on the text.

We are grateful to the US National Science Foundation, the Linguistic Data Consortium, and the Universities of Pennsylvania, Edinburgh, and Melbourne for supporting our work on this book.

About the Authors

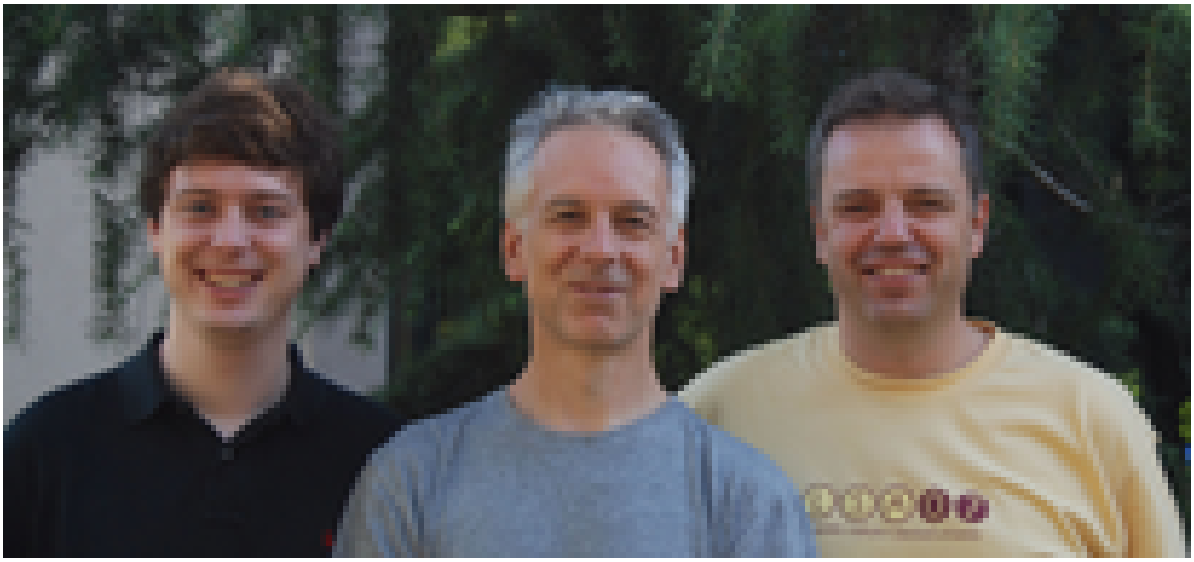

Figure 1: Edward Loper, Ewan Klein, and Steven Bird, Stanford, July 2007

Steven Bird is Associate Professor in the Department of Computer Science and Software Engineering at the University of Melbourne, and Senior Research Associate in the Linguistic Data Consortium at the University of Pennsylvania. After completing his undergraduate training in computer science and mathematics at the University of Melbourne, Steven went to the University of Edinburgh to study computational linguistics, and completed his PhD in 1990 under the supervision of Ewan Klein. He later moved to Cameroon to conduct linguistic fieldwork on the Grassfields Bantu languages under the auspices of the Summer Institute of Linguistics. More recently, he spent several years as Associate Director of the Linguistic Data Consortium where he led an R&D team to create models and tools for large databases of annotated text. Back at Melbourne University, he established a language technology

research group and has taught at all levels of the undergraduate computer science curriculum. Steven is Vice President of the Association for Computational Linguistics.

Ewan Klein is Professor of Language Technology in the School of Informatics at the University of Edinburgh. He completed a PhD on formal semantics at the University of Cambridge in 1978. After some years working at the Universities of Sussex and Newcastle upon Tyne, Ewan took up a teaching position at Edinburgh. He was involved in the establishment of Edinburgh's Language Technology Group 1993, and has been closely associated with it ever since. From 2000–2002, he took leave from the University to act as Research Manager for the Edinburgh-based Natural Language Research Group of Edify Corporation, Santa Clara, and was responsible for spoken dialogue processing. Ewan is a past President of the European Chapter of the Association for Computational Linguistics and was a founding member and Coordinator of the European Network of Excellence in Human Language Technologies (ELSNET). He has been involved in leading numerous academic-industrial collaborative projects, the most recent of which is a biological text mining initiative funded by ITI Life Sciences, Scotland, in collaboration with Cognia Corporation, NY.

Edward Loper is a doctoral student in the Department of Computer and Information Sciences at the University of Pennsylvania, conducting research on machine learning in natural language processing. Edward was a student in Steven's graduate course on computational linguistics in the fall of 2000, and went on to be a TA and share in the development of NLTK. In addition to NLTK, he has helped develop other major packages for documenting and testing Python software, `epydoc` and `doctest`.

About this document...