# NLTK:
# The Natural Language Toolkit

Edward Loper

# Natural Language Processing

- Use computational methods to process human language.
- Examples:
  - Machine translation
  - Text classification
  - Text summarization
  - Question answering
  - Natural language interfaces

# Teaching NLP

- How do you create a strong practical component for an introductory NLP course?
  - Students come from diverse backgrounds (CS, linguistics, cognitive science, etc.)
    - Many students are learning to program for the first time.
    - We want to teach NLP, not programming.
  - Processing natural language can involve lots of low-level "house-keeping" tasks
    - Not enough time left to learn the subject matter itself.
  - Diverse subject matter

# NLTK: Python-Based NLP Courseware

- NLTK: Natural Language Toolkit
  - A suite of Python packages, tutorials, problem sets, and reference documentation.
  - Provides standard data types and interfaces for NLP tasks.
- Development:
  - Created during a graduate NLP course at U. Penn (2001)
  - Extended & redesigned during subsequent semesters.
  - Many additions from student projects & outside contributors.
- Deployment:
  - Released under GPL (code) and creative commons (docs).
  - Used for teaching intro NLP at 8 universities
  - Used by students & researchers for independent study
- http://nltk.sourceforge.net

# NLTK Uses

- **Course Assignments:**
  - **Use an existing module to explore an algorithm or perform an experiment.**
  - **Combine modules to form a complete system.**
- **Class demonstrations:**
  - **Tedious algorithms come to life with online demonstrations.**
  - **Interactive demos allow live topic exploration.**
- **Advanced Projects:**
  - **Implement new algorithms.**
  - **Add new functionality.**

# Design Goals

## Requirements

- Ease of use
- Consistency
- Extensibility
- Documentation
- Simplicity
- Modularity

## Non-requirements

- Comprehensiveness
- Efficiency
- Cleverness

# Why Use Python?

- Shallow learning curve
- Python code is exceptionally readable
    - “Executable pseudocode”
- Interpreted language
    - Interactive exploration
    - Immediate feedback
- Extensive standard library
- Light-weight object oriented system
    - Useful when it’s needed
    - But doesn’t get in the way when it’s not
- Generators make it easy to demonstrate algorithms
    - More on this later.

# Design Overview

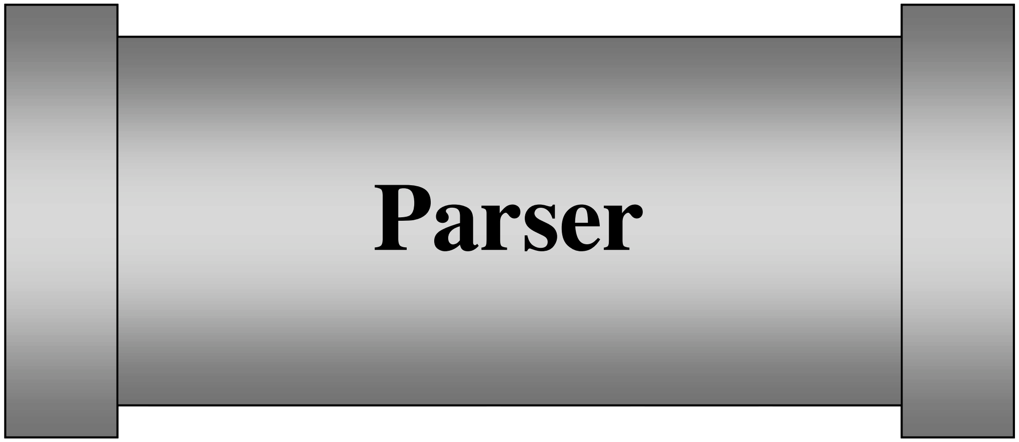
- **Flow control is organized around NLP *tasks*.**
    - **Examples: tokenizing, tagging, parsing**
- **Each task is defined by an *interface*.**
    - **Implemented as a stub base class with docstrings**
- **Multiple *implementations* of each task.**
    - **Different techniques and algorithms**
    - **Different algorithms**
- **Tasks communicate using a standard data type:**
    - **The `Token` class.**

# Pipelines and Blackboards

- Traditionally, NLP processing is described using a transformational model: *“The pipeline”*
  - A series of pipeline stages transforms information.

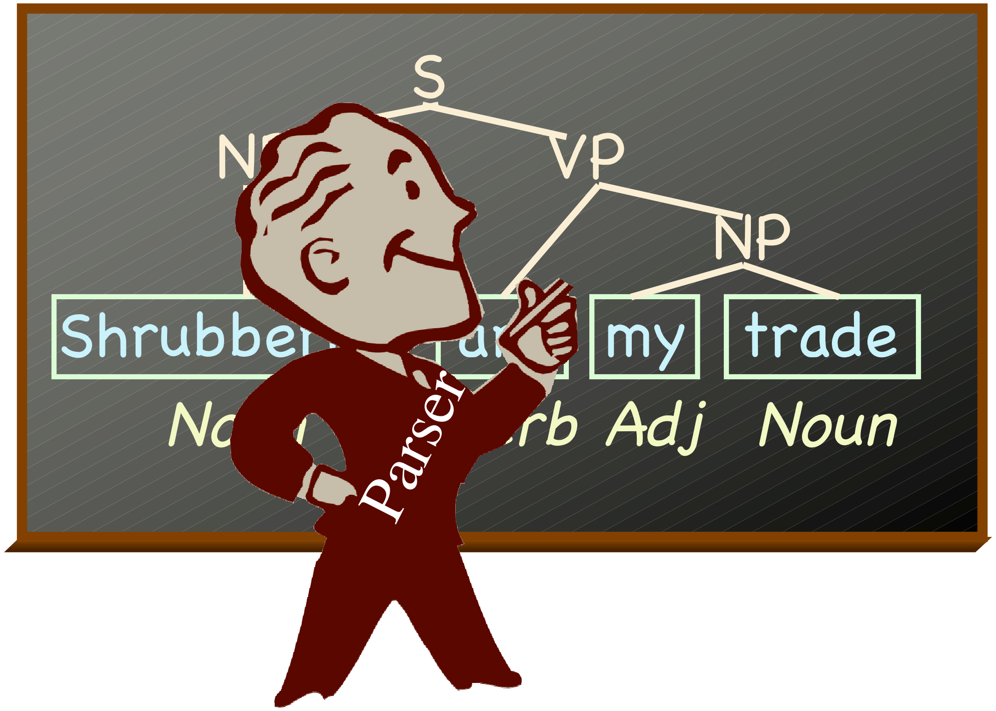
- For an educational toolkit, we prefer to use an annotation-based model: “The blackboard”
  - A series of annotators add information.

# The Pipeline Model



- A series of sequential transformations.
- Input format ≠ Output format.
- Only preserve the information you need.

# The Blackboard Model



- Task process a single shared data structure
- Each task adds new information

# Advantages of the Blackboard

- Easier to experiment
  - Tasks can be easily rearranged.
  - Students can swap in new implementations that have different requirements.
  - No need to worry about “threading” info through the system.
- Easier to debug
  - We don’t throw anything away.
- Easier to understand
  - We build a single unified picture.

# Tokens

- Represent individual pieces of language.
  - E.g., documents, sentences, and words.
- Each token consists of a set of properties:
  - Each property maps a name to a value.
- Some typical properties:

| | | | |
|---|---|---|---|
| *TEXT* | Text content | *WAVE* | Audio content |
| *POS* | Part of speech | *SENSE* | Word sense |
| *TREE* | Parse tree | *WORDS* | Contained words |
| *STEM* | Word stem | | |

# Properties

- **Properties are not fixed or predefined.**
  - **Consenting adults.**
  - **Dynamic polymorphism.**
- **Properties are mutable.**
  - **But typically mutated *monotonically*. I.e., only add properties; don’t delete or modify them.**
- **Properties can contain/point to other tokens.**
  - **A sentence token’s *WORDS* property**
  - **A tree token’s *PARENT* property.**

# Locations:
# Unique Identifiers for Tokens

- **How many words in this phrase?**

*An African swallow or a European swallow.*

**a) 5 b) 6 c) 7 d) 8**

# Locations:
# Unique Identifiers for Tokens

- **How many words in this phrase?**

*1 2 3 4 5 6 7*

*An African swallow or a European swallow*

**a) 5   b) 6   [✓ c) 7]   d) 8**

1. **An**
2. **African**
3. **swallow**
4. **or**
5. **a**
6. **European**
7. **swallow**

# Locations:
# Unique Identifiers for Tokens

- **How many words in this phrase?**

*1 2 3 4 5 6 3*
*An African swallow or a European swallow*

**a) 5** **[✓ b) 6]** **c) 7** **d) 8**

1. **An**
2. **African**
3. **swallow**
4. **or**
5. **a**
6. **European**

# Locations:
# Unique Identifiers for Tokens

- **How many words in this phrase?**

*An African swallow or a European swallow*

- **Need to distinguish between an abstract piece of language and an occurrence.**
- **Create unique identifiers for Tokens**
  - **Based on their locations in the containing text.**
  - **Stored in the *LOC* property**

# Specialized Tokens

- **Use subclasses of Token to add specialized behavior.**
- **E.g.,** `ParentedTreeToken` **adds...**
  - **Standard tree operations.**
    - `height()`, `leaves()`, **etc.**
  - **Automatically maintained parent pointers.**
- **All data is stored in properties.**

# Task Interfaces

- **Each task is defined by an *interface*.**
  - **Implemented as a stub base class with docstrings.**
  - **Conventionally named with a trailing “I”**
  - **Used only for documentation purposes.**
- **All interfaces have the same basic form:**
  - **An “action” method monotonically mutates a token.**

```
class ParserI:
   def parse(token):
       """
       A processing class for deriving trees that …
       """
```

# Variations on a Theme

- **Where appropriate, interfaces can define a set of extended action methods:**
  - **`action()`** **The basic action method.**
  - **`action_n()`** **A variant that outputs the *n* best solutions.**
  - **`action_dist()`** **A variant that outputs a probability distribution over solutions.**
  - **`xaction()`** **A variant that consumes and generates iterators.**
  - **`raw_action()`** **A transformational (pipeline) variant.**

# Building Algorithm Demos

- An example algorithm: CKY

```
for w in range(2, N):
    for i in range(N-w):
        for  k in range(1, w-1):
            if $A \rightarrow BC$ and $B \rightarrow \alpha \in$chart[i][i+k] and $C \rightarrow \beta \in$chart[i+k][i+w]:
                chart[i][i+w].append(A→BC)
```

- How do we build an interactive GUI demo?
  - Students should be able to see each step.
  - Students should be able to tweak the algorithm

# Building Algorithm Demos: Generators to the Rescue!

- A generator is a resumable function.
- Add a `yield` to stop the algorithm after each step.

```
for w in range(2, N):
    for i in range(N-w):
        for  k in range(1, w-1):
            if $A \rightarrow BC$ and $B \rightarrow \alpha \in$ chart[i][i+k] and $C \rightarrow \beta \in$ chart[i+k][i+w]:
                chart[i][i+w].append($A \rightarrow BC$)
                yield A →BC
```

- Accessing algorithm state:
  - Yield a value describing the state or the change
  - Use member variables to store state (`self.chart`)

# Example: Parsing

- What is it like to teach a course using NLTK?
- Demonstration:
  - Two kinds of parsing
  - Two ways to use NLTK

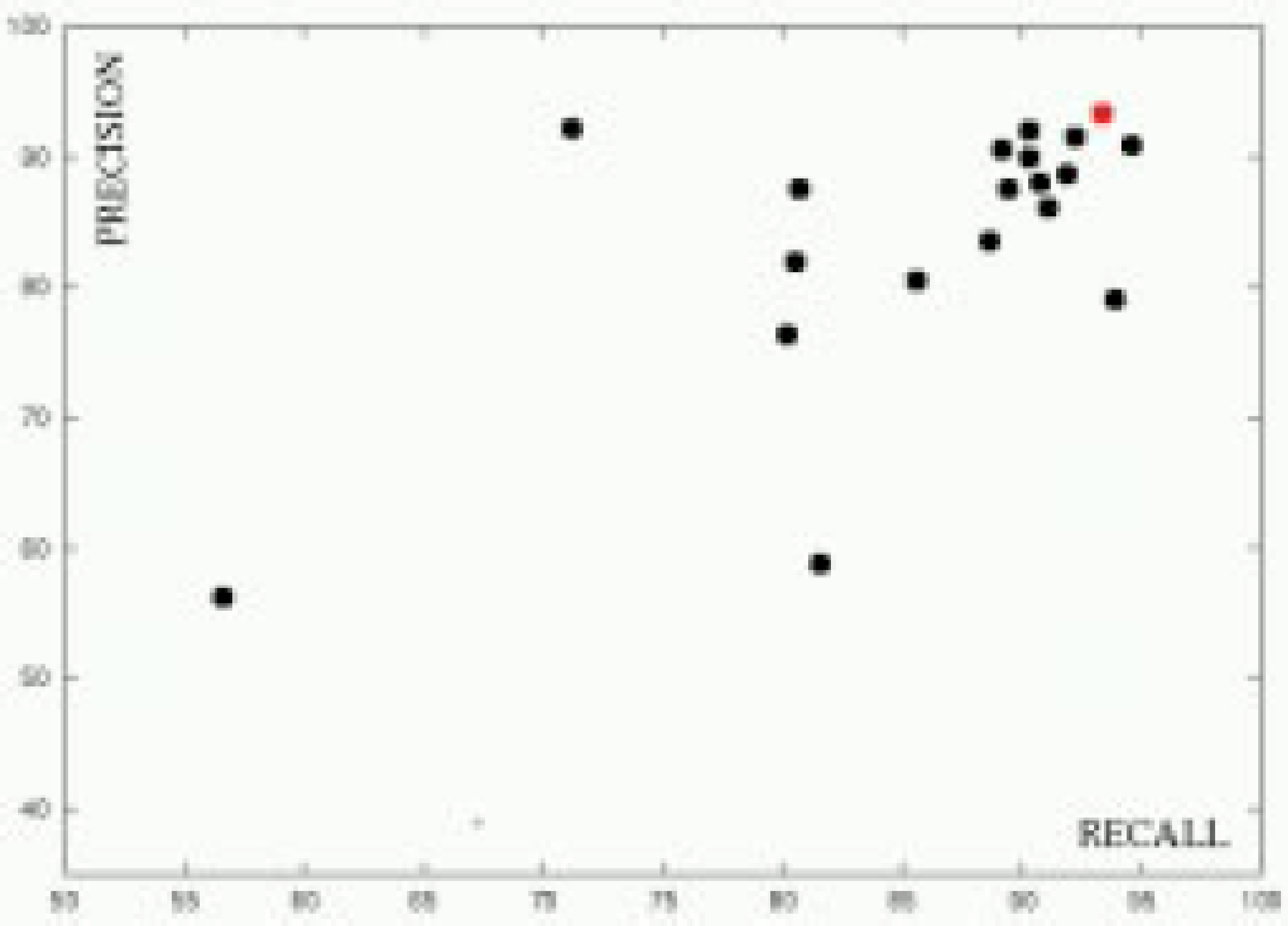
A) Assignments: chunk parsing

B) Demonstrations: chart parsing

# Chunk Parsing

- Basic task:
    - Find the noun phrases in a sentence.
- Students were given…
    - A regular-expression based chunk parser
    - A large corpus of tagged text
- Students were asked to…
    - Create a cascade of chunk rules
    - Use those rules to build a chunk parser
    - Evaluate their system's performance

# Competition Scoring



# Chart Parsing

- **Basic task:**
  - **Find the structure of a sentence.**
- **Chart parsing:**
  - **An efficient parsing algorithm.**
  - **Based on dynamic programming.**
    - **Store partial results, so we don’t have to recalculate them.**
- **Chart parsing demo:**
  - **Used for live in-class demonstrations.**
  - **Used for at-home exploration of the algorithm.**

# Conclusions

- Some lessons learned:
  - Use simple & flexible inter-task communication
    - A general polymorphic data type
    - Simple standard interfaces
  - Use blackboards, not pipelines.
  - Don’t throw anything away unless you have to.
  - Generators are a great way to demonstrate algorithms.

# Natural Language Toolkit

- If you're interested in learning more about NLP, we encourage you to try out the toolkit.
- If you are interested in contributing to NLTK, or have ideas for improvement, please contact us.
- Open session: today at 2:15 (Room 307)

| | |
|---|---|
| URL: | http://nltk.sf.net |
| Email: | ed@loper.org |
| | sb@unagi.cis.upenn.edu |