Wednesday, December 3
Announcements

• Homework 10 due Wednesday 12/3 @ 11:59pm
• Quiz 3 released Wednesday, due Thursday 12/4 @ 11:59pm
• No videos for Lecture 38 on Friday 12/5
  ▪ Come to class and take the final survey
  ▪ There will be a screencast of live lecture (http://goo.gl/hyUTca)
• Final exam held on Thursday 12/18 3pm–6pm
  ▪ 30 hours of review sessions next week! Monday – Friday 11am–6pm (mostly in 271 Soda)
Ambiguity
Programs must be written for people to read\(^1\)

\(^1\)Preface of *Structure and Interpretation of Computer Programs* by Harold Abelson and Gerald Sussman with Julie Sussman
Programs must be written for people to read

Preface of *Structure and Interpretation of Computer Programs* by Harold Abelson and Gerald Sussman with Julie Sussman
Syntactic Ambiguity in English

Programs must be written for people to read

1Preface of *Structure and Interpretation of Computer Programs* by Harold Abelson and Gerald Sussman with Julie Sussman
Syntactic Ambiguity in English

**program** (noun)
- a series of coded software instructions

**program** (verb)
- provide a computer with coded instructions

**must** (verb)
- be obliged to

**must** (noun)
- dampness or mold

Programs must be written for people to read.
Syntax Trees
Representing Syntactic Structure

A **Tree** represents a phrase:
- **tag** — What kind of phrase (e.g., S, NP, VP)
- **branches** — Sequence of Tree or Leaf components

A **Leaf** represents a single word:
- **tag** — What kind of word (e.g., N, V)
- **word** — The word

```
beasts = Leaf('N', 'buffalo')
intimidate = Leaf('V', 'buffalo')
S, NP, VP = 'S', 'NP', 'VP'
Tree(S, [Tree(NP, [beasts]),
       Tree(VP, [intimidate,
                Tree(NP, [beasts])])])
```
Grammars
Context-Free Grammar Rules

A grammar rule describes how a tag can be expanded as a sequence of tags or words.

A Sentence ...

... can be expanded as ...

... a Noun Phrase then a Verb Phrase.

<table>
<thead>
<tr>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
</tr>
<tr>
<td>NP → N</td>
</tr>
<tr>
<td>N → buffalo</td>
</tr>
<tr>
<td>VP → V NP</td>
</tr>
<tr>
<td>V → buffalo</td>
</tr>
</tbody>
</table>

(Demo)
Parsing
Exhaustive Parsing

Expand all tags recursively, but constrain words to match input

```
S

NP  VP
```

```
0  buffalo  1  buffalo  2  buffalo  3  buffalo  4
```
Exhaustive Parsing

Expand all tags recursively, but constrain words to match input

Constraint: A Leaf must match the input word

buffalo buffalo buffalo buffalo
Exhaustive Parsing

Expand all tags recursively, but constrain words to match input

```
  S
/   \   
NP   VP

0 1 2 3 4 buffalo buffalo buffalo buffalo
```
Exhaustive Parsing

Expand all tags recursively, but constrain words to match input
Learning

(Demo)
Scoring a Tree Using Relative Frequencies

Not all syntactic structures are equally common

```
S → NP VP
NP → NN NNS
VP → VB NP
NP → NNS
```

```
  NN   NNS   VB   NNS
```

teacher strikes idle kids

**Rule frequency per 100,000 tags**

- S → NP VP: 25372
- NP → NN NNS: 1335
- VP → VB NP: 6679
- NP → NNS: 4282
- NN → teacher: 5
- NNS → strikes: 25
- VB → idle: 26
- NNS → kids: 32
Scoring a Tree Using Relative Frequencies

Not all syntactic structures are equally common

```
S → NP VP
```

```
NP → NN
```

```
VP → VBZ NP
```

```
NP → JJ NNS
```

**teacher strikes idle kids**

**Rule frequency per 100,000 tags**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Frequency 1</th>
<th>Frequency 2</th>
<th>Rule</th>
<th>Frequency 1</th>
<th>Frequency 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>25372</td>
<td></td>
<td>NN → teacher</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>NP → NN</td>
<td>1335</td>
<td>4358</td>
<td>VBZ → strikes</td>
<td>25</td>
<td>19</td>
</tr>
<tr>
<td>VP → VBZ NP</td>
<td>6679</td>
<td>3160</td>
<td>JJ → idle</td>
<td>26</td>
<td>18</td>
</tr>
<tr>
<td>NP → JJ NNS</td>
<td>4282</td>
<td>2526</td>
<td>NNS → kids</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

(Demo)