Syntax
Class outline:

- Syntax trees
- Data abstractions
- Parsing syntax trees
- Sentence generation
Syntax trees
Syntax trees

Both programming languages and spoken languages can be parsed into syntax trees.

For a spoken language, a syntax tree reveals the syntactic structure of a single sentence.

"This is a book"
Syntax tree terminals

The leaves are also called **terminals**: they contain both a syntactic identifier (**tag**) and the actual world.

- **NN**: singular noun (e.g. "This", "book")
- **COP**: copula (e.g. "is")
- **DT**: determiner (e.g. "the")

Other terminals: **NNS** (plural noun), **NNP** (proper noun), **PRP** (personal pronoun), **JJ** (adjective), **IN** (preposition), **CC** (coordinating conjunction), **AUX** (auxillary verb), **RB** (adverb), **VBN** (verb, past participle), ...
Syntax tree non-terminals

The other nodes are called **non-terminals** and contain only tags (typically a phrase type). The tag describes the phrase in the leaves under them.

- **S**: sentence (e.g. "This is a book")
- **NP**: noun phrase (e.g. "This", "a book")
- **VP**: verb phrase (e.g. "is a book")

Other non-terminals: **SQ** (question), **PP** (prepositional
More syntax trees

"Is that a big bug or a little bug?"
More syntax trees

"I've never seen such a cute kangaroo."
Syntax tree representation
Using the tree abstraction

The label of non-terminals will be just the tag: "S", "NP", "VP".

The label of terminals will be a list of the tag and the word itself: ["NN", "This"], ["COP", "is"], ["DT", "a"], ["NN", "book"].
A tree() version

t = tree("S", [
    tree("NP", [tree(["NN", "this"])]),
    tree("VP", [
        tree(["COP", "is"],
        tree("NP", [
            tree(["DT", "a"],
            tree(["NN", "book"])
        ])])
    ])
  )
}
def phrase(tag, branches):
    return tree(tag, branches)

def word(tag, text):
    return tree([tag, text])

def text(word):
    return label(word)[1]

def tag(t):
    """Return the tag of a phrase or word.""
    if is_leaf(t):
        return label(t)[0]
    else:
        return label(t)
Parsing
Writing files into trees

Input data: 

- suppes.parsed

```plaintext
(ROOT (S (NP (NN this)) (VP (COP is) (NP (DT a) (NN book)))) (. .))

(ROOT (S (NP (PRP I)))
  (VP (AUX 've)
   (ADVP (RB never))
   (VP (VBN seen) (NP (DT such) (DT a) (JJ cute) (NN kangaroo))))
  (. .))
```

Desired output: 

- tree()s!

File comes from:
Reading files in Python

Here are two ways to read a plain text file:

```python
open('/some/file.txt').read()
```

A list of strings, each containing one line:

```python
open('/some/file.txt').readlines()
```

Using `readlines()` on the input file:

```python
open('suppes.parse').readlines()
```
Useful string methods

`str.strip()` returns a string without whitespace (spaces, tabs, etc.) on the ends

```
' hello ' .strip()
```

`str.split(sep=None)` returns a list of strings that were separated by `sep`

```
'hi  there ' .split()
```

`str.replace(a, b)` returns a string with all instances of string `a` replaced by string `b`

```
'2+2'.replace( '+', ' + ')
```
Useful string methods

- `str.strip()` returns a string without whitespace (spaces, tabs, etc.) on the ends

```
'hello '.strip()  # 'hello'
```

- `str.split(sep=None)` returns a list of strings that were separated by `sep`

```
'hi there '.split()
```

- `str.replace(a, b)` returns a string with all instances of string `a` replaced by string `b`

```
'2+2'.replace('+', ' + ')
```
Useful string methods

```
str.strip() returns a string without whitespace (spaces, tabs, etc.) on the ends

' hello '.strip()  # 'hello'

str.split(sep=None) returns a list of strings that were separated by sep

'hi there '.split()  # ['hi', 'there']

str.replace(a, b) returns a string with all instances of string a replaced by string b

'2+2'.replace('+', ' + ')
```
Useful string methods

```python
str.strip() returns a string without whitespace (spaces, tabs, etc.) on the ends

'hello '.strip()  # 'hello'
```

```python
str.split(sep=None) returns a list of strings that were separated by sep

'hi there '.split()  # ['hi', 'there']
```

```python
str.replace(a, b) returns a string with all instances of string a replaced by string b

'2+2'.replace('+', ' + ')  # '2  + 2'
```
From lines to tokens

```
['(ROOT (S (NP (NN this)) (VP (COP is) (NP (DT a) (NN book)))) (. ?))
    \n',..]
```

to

```
[['(','ROOT', ',', 'S', ',', 'NP', ',', 'NN', ',','this', ',', '),'),
    ('(', 'VP', ',', 'COP', ',', 'is', ',', ')'),
    ('(', 'NP', ',', 'DT', ',', 'a', ',', ')
    ('(', 'NN', ',', 'book', ',', ')'),
    '),
    ('(', '.', ',','?', ',', ')'),
    ')]
```  

**read_sentences** takes care of this:

```python
lines = open('suppesparsed').readlines()
tokens = read_sentences(lines)
```
From tokens to trees

```python
[...,
'(', 'NP', '(', 'DT', 'a', ')', '(', 'JJ', 'big', ')', '(',
    # i

def read_parse_tree(tokens, i):
    # Read the tag, which is tokens[i], then advance i.
    # While the current item is a '(',
    # call read_parse_tree to construct a branch.
    # Once the current item is a ')',
    # return a phrase from the tag and branches.
    # Base case: there is no '(' or ')'
    # because there is just text after the tag.

    tree = read_parse_tree(tokens[0], 1)
```

**read_parse_tree** will return the tree it read and what to read next.
Generating sentences
Language models

A statistical (or probabilistic) language model describes how likely some text would be.

What word do you think appears at the end of this ____?
Language models

A statistical (or probabilistic) language model describes how likely some text would be.

**Sampling** from a statistical language model uses that description to generate language.

A useful language model needs to generalize from examples.
E.g., Substitute any phrase in an utterance with any other phrase that has the same tag.

- (S (NP (DT the) (NN dog)) (VP (VBD ran)))
- (S (NP (DT the) (NN water)) (VP (VBD evaporated)))
def index_trees(trees):
    """Return a dictionary from tags to lists of trees."""
    index = {}
    for t in trees:
        for tag, node in nodes(t):
            if tag not in index:
                index[tag] = []
            index[tag].append(node)
    return index

trees = [tokens_to_parse_tree(s) for s in all_sentences()]
tree_index = index_trees(trees)
Generating new trees

Then we need a sampling strategy:

- Starting with the branches of the root node, flip a coin for each branch.
- If it comes up heads, swap that branch for another branch (phrase or word) that has the same tag.
- Then, apply this procedure to all of the branches.

```python
def gen_tree(t, tree_index, flip):
    """Return a version of t in which branches are randomly replaced."""
    new_branches = []
    if is_leaf(t):
        return t
    for b in branches(t):
        if flip():
            b = random.choice(tree_index[tag(b)])
            new_branches.append(gen_tree(b, tree_index, flip))
    return phrase(tag(t), new_branches)
```
Python Project of The Day!
Natural Language Toolkit

**NLTK**: An open-source Python library for language modeling, spelling correction, text classification, sentiment analysis, information retrieval, relation extraction, recommendation systems, translation question answering, word vectors, and more.

**Demo: Sentence trees!**

Further learning: Github repo, NLTK Book, NLTK Sentiment Analysis