[Natural Language Processing](https://www.codecademy.com/learn/natural-language-processing)

Connected to Codecademy

**Intro to NLP**

Look at the technologies around us:

* Spellcheck and autocorrect
* Auto-generated video captions
* Virtual assistants like Amazon’s Alexa
* Autocomplete
* Your news site’s suggested articles

What do they have in common?NLP got its start around 1950 with Alan Turing’s test for artificial intelligence evaluating whether a computer can use language to fool humans into believing it’s human.

But approximating human speech is only one of a wide range of applications for NLP! Applications from detecting spam emails or bias in tweets to improving accessibility for people with disabilities all rely heavily on natural language processing techniques.

NLP can be conducted in several programming languages. However, Python has some of the most extensive open-source NLP libraries, including the [Natural Language Toolkit](https://www.nltk.org/) or ***NLTK***.

Preprocessing

Parsing

[Natural Language Processing](https://www.codecademy.com/learn/natural-language-processing)

Connected to Codecademy

**Text Preprocessing**

Without preprocessing, your computer interprets "the", "The", and "<p>The" as entirely different words. There is a LOT you can do here, depending on the formatting you need. Lucky for you, [Regex](https://en.wikipedia.org/wiki/Regular_expression" \t "_blank) and NLTK will do most of it for you! Common tasks include:

**Noise removal** — stripping text of formatting (e.g., HTML tags).

**Tokenization** — breaking text into individual words.

**Normalization** — cleaning text data in any other way:

* **Stemming** is a blunt axe to chop off word prefixes and suffixes. “booing” and “booed” become “boo”, but “sing” may become “s” and “sung” would remain “sung.”
* **Lemmatization** is a scalpel to bring words down to their root forms. For example, NLTK’s savvy lemmatizer knows “am” and “are” are related to “be.”
* Other common tasks include lowercasing, punctuation removal, [stopwords](https://en.wikipedia.org/wiki/Stop_words" \t "_blank) removal, spelling correction, etc.

Why are the lemmatized verbs like "went" still conjugated? By default lemmatize() treats every word as a noun.

Give lemmatize a second argument: get\_part\_of\_speech(token). This will tell our lemmatizer what part of speech the word is.

**Parsing Text**

You now have a preprocessed, clean list of words. Now what? It may be helpful to know how the words relate to each other and the underlying syntax (grammar). ***Parsing*** is a stage of NLP concerned with segmenting text based on syntax.

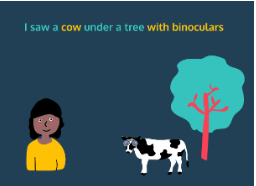
You probably do not want to be doing any parsing by hand and NLTK has a few tricks up its sleeve to help you out:

***Part-of-speech tagging (POS tagging)*** identifies parts of speech (verbs, nouns, adjectives, etc.). NLTK can do it faster (and maybe more accurately) than your grammar teacher!

***Named entity recognition (NER)*** helps identify the proper nouns (e.g., “Natalia” or “Berlin”) in a text. This can be a clue as to the topic of the text and NLTK captures many for you.

***Dependency grammar*** trees help you understand the relationship between the words in a sentence. It can be a tedious task for a human, so the Python library spaCy is at your service, even if it isn’t always perfect.

In English we leave a lot of ambiguity, so syntax can be tough, even for a computer program. Take a look at the following sentence:



Do I have the binoculars? Does the cow have binoculars? Does the tree have binoculars?

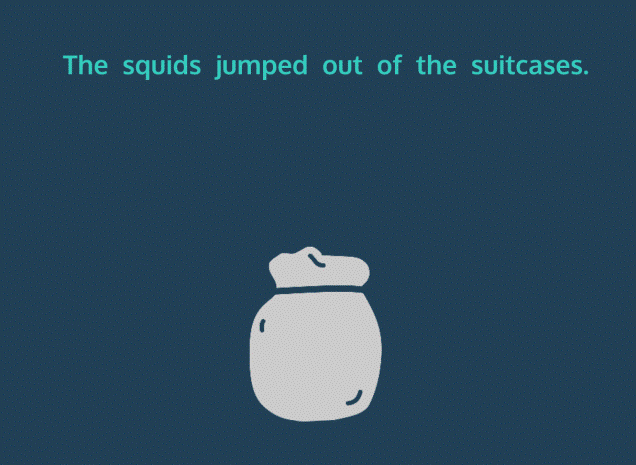
***Regex parsing***, using Python’s re library, allows for a bit more nuance. When coupled with POS tagging, you can identify specific phrase chunks. On its own, it can find you addresses, emails, and many other common patterns within large chunks of text.

**Language Models - Bag-of-Words Approach**

We can help computers make predictions about language by training a language model on a *corpus* (a bunch of example text).

***Language models*** are probabilistic computer models of language. We build and use these models to figure out the likelihood that a given sound, letter, word, or phrase will be used. Once a model has been trained, it can be tested out on new texts.

One of the most common language models is the unigram model, a statistical language model commonly known as ***bag-of-words***. As its name suggests, bag-of-words does not have much order to its chaos! What it does have is a tally count of each instance for each word. Consider the following text example:



Provided some initial preprocessing, bag-of-words would result in a mapping like:

{"the": 2, "squid": 1, "jump": 1, "out": 1, "of": 1, "suitcase": 1}

Now look at this sentence and mapping: “Why are your suitcases full of jumping squids?”

{"why": 1, "be": 1, "your": 1, "suitcase": 1, "full": 1, "of": 1, "jump": 1, "squid": 1}

You can see how even with different word order and sentence structures, “jump,” “squid,” and “suitcase” are shared topics between the two examples. Bag-of-words can be an excellent way of looking at language when you want to make predictions concerning topic or sentiment of a text. When grammar and word order are irrelevant, this is probably a good model to use.

# Language Models - N-Grams and NLM

For parsing entire phrases or conducting language prediction, you will want to use a model that pays attention to each word’s neighbors. **n-gram** model considers a sequence of some number (n) units and calculates the probability of each unit in a body of language given the preceding sequence of length n. Through this, n-gram probabilities with larger n values can be useful in language prediction.

Take a look at our revised squid example: “The squids jumped out of the suitcases. The squids were furious.”

A bigram model (where n is 2) might give us the following count frequencies:

{('', 'the'): 2, ('the', 'squids'): 2, ('squids', 'jumped'): 1, ('jumped', 'out'): 1, ('out', 'of'): 1, ('of', 'the'): 1, ('the', 'suitcases'): 1, ('suitcases', ''): 1, ('squids', 'were'): 1, ('were', 'furious'): 1, ('furious', ''): 1}

There are a couple problems with the n gram model:

1. How can your language model make sense of the sentence “The cat fell asleep in the mailbox” if it’s never seen the word “mailbox” before? During training, your model will probably come across test words that it has never encountered before (this issue also pertains to bag of words). A tactic known as language smoothing can help adjust probabilities for unknown words, but it isn’t always ideal.
2. For a model that more accurately predicts human language patterns, you want n (your sequence length) to be as large as possible. That way, you will have more natural sounding language, right? Well, as the sequence length grows, the number of examples of each sequence within your training corpus shrinks. With too few examples, you won’t have enough data to make many predictions.

Enter **neural language models (NLM)!** Much recent work within NLP has involved developing and training neural networks to approximate the approach our human brains take towards language. This deep learning approach allows computers a much more adaptive tack to processing human language.

Dispatch me. What have you? I am but a willing and capable participant in the founding of a new program. But you are not sure. You are not keen to predict upon my capacities a favorable outcome without first a sort of favorable marks.