# Markov text generation

**Write a bare-bones Markov text generator.**

Implement a function of the form

```
finish_sentence(sentence, n, corpus, randomize=False)
```

that takes four arguments:

1. a `sentence` [tuple of tokens] that we're trying to build on,
2. `n` [int], the length of n-grams to use for prediction, and
3. a source `corpus` [tuple of tokens]
4. a flag indicating whether the process should be `randomize` [bool]

and returns an extended sentence until the first `.`, `?`, or `!` is found OR until it has 10 total tokens.

If the input flag `randomize` is false, the text generator should be deterministic. Choose at each step the single most probable next token. When multiple tokens are equally probable, choose the one that is first alphabetically.

If `randomize` is true, draw the next word randomly from the appropriate distribution.

Use stupid backoff ($\alpha = 0.4$) and no smoothing.

**Provide some example applications of your function in both deterministic and stochastic modes, for a few sets of seed words and a few different `n`.**

As one (simple) test case, use the following inputs:

```
sentence = ['she', 'was', 'not']
n = 3
corpus = nltk.word_tokenize(
    nltk.corpus.gutenberg.raw('austen-sense.txt').lower()
)
randomize = False
```

For this deterministic case, the result should be:

```
['she', 'was', 'not', 'in', 'the', 'world', ',', 'and', 'the', 'two']
```

Add your method to a file/module named `mtg.py` and use the test script `test_mtg.py` to verify that the test examples work.

You should turn in a document (`.txt`, `.md`, or `.pdf`) answering all of the **red** items above. You should also turn in Python scripts (`.py`) for *each* of the **blue** items. Unless otherwise specified, you may use only `numpy` and the standard library.