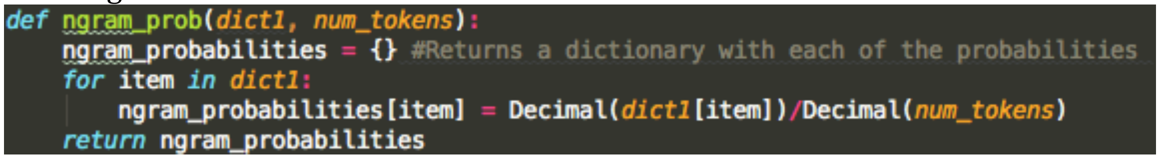
Sean McNeil (srm274) and Molly Higgins (mfh66)
Project1 Part 1

Our team completed step 1 and 2 or project 1 using Python 2.7.

Step 1:

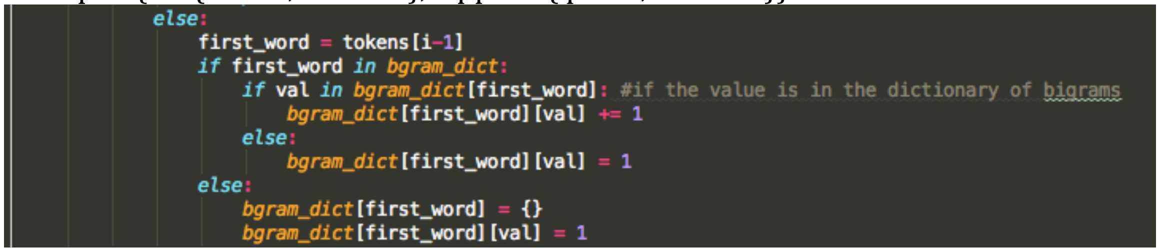
We began our project with writing our own tokenizer. Our tokenzier converts everything to lowercase and takes punctuation into account. We tried to implement NLTK, but it did not work as well because it had trouble with Unicode.

For our unigram model, we have two dictionaries. One dictionary contains the unigram (key) and the number of occurrences (value). The second dictionary contains the unigram (key) and the probability it will occur (value). This probability is calculated using the total number of unigrams in the corpus and the number times this unigram occurs.



```
def ngram_prob(dict1, num_tokens):
    ngram_probabilities = {} #Returns a dictionary with each of the probabilities
    for item in dict1:
        ngram_probabilities[item] = Decimal(dict1[item])/Decimal(num_tokens)
    return ngram_probabilities
```

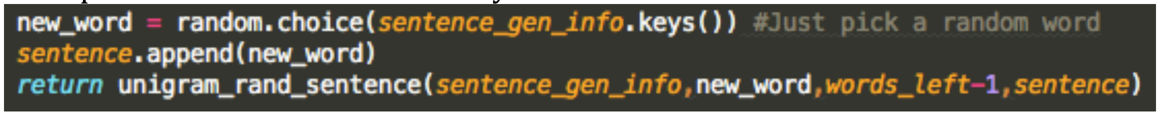
For our bigrams model, we have three data structures. The first data structure, like for our unigram model, is a dictionary containing all of the bigrams (key) and the number of occurrences (value). The second dictionary, like our unigram model, is a dictionary with the probability each bigram will occur, given that the first word occurs. This is the number of times the first word occurs/number of times this word comes after it. The third data structure is a dictionary where the keys consist of the 1st word of the bigram and the value pair consists of a second dictionary. This second dictionary has keys that consist of the 2nd word of the bigram and value pairs that consist of the number of occurrences of that pair. For example: {'a': {'car':1, 'bike': 2}, 'apple' : {'pie: 1, 'tree' : 3}}.



```
            else:
                first_word = tokens[i-1]
                if first_word in bgram_dict:
                    if val in bgram_dict[first_word]: #if the value is in the dictionary of bigrams
                        bgram_dict[first_word][val] += 1
                    else:
                        bgram_dict[first_word][val] = 1
                else:
                    bgram_dict[first_word] = {}
                    bgram_dict[first_word][val] = 1
```

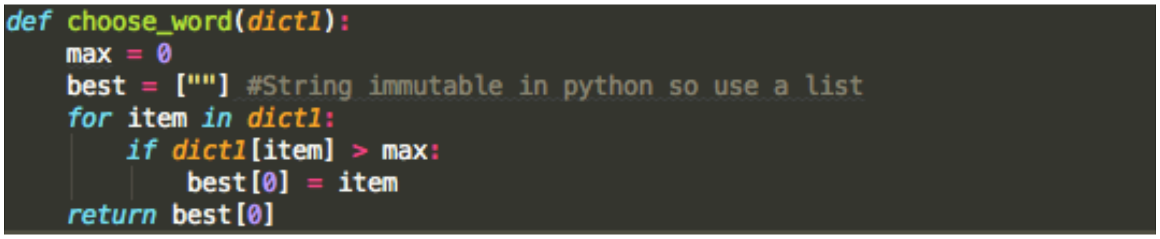
Step 2:

To generate random sentences for unigrams, we choose words randomly. We did this because if we chose the highest probability unigrams each time, we would end up with the same sentence every time.



```
new_word = random.choice(sentence_gen_info.keys()) #Just pick a random word
sentence.append(new_word)
return unigram_rand_sentence(sentence_gen_info,new_word,words_left-1,sentence)
```

To generate random sentences for bigrams, we chose words based on probabilities. We chose the next word based on the highest probabilities in the dictionary.



```python
def choose_word(dict1):
    max = 0
    best = [""] #String immutable in python so use a list
    for item in dict1:
        if dict1[item] > max:
            best[0] = item
    return best[0]
```

We chose to allow the user to dictate the length of the random sentence they would like to generate. If the sentence only terminates with punctuation, the generator could end up looping on certain words forever, or ending only 2 words in. Our code automatically adds a period to sentences.

Random Sentences

Genre	Ngram	Length	Sentence
Children	Bigram	10	Downy silver could observe if anyone gave order came careering toward.
Children	Unigram	12	Monuments prevailing affording routine breathe vexes bedizened soup tables howf unsightly novelty.
Crime	Unigram	8	Monograph businesslike renounced signet blight exquisitely right_ beloved demeanor.
Crime	Bigram	18	Persuade ourselves the bereaved family silver of industry it once furnish the bereaved family silver or indirectly it once.
History	Unigram	9	Cavil engagement dart physical sidicini _*Ephesus*_ 1806 athamanes honoured shipwreck.
History	Bigram	16	Warfare dated back the 8th _*virilis*_, augmented that once asked air came suddenly at once.

Notes on random sentences:

Because our random sentences for unigrams are generated completely randomly, uncommon words appear much more than in a normal sentence.
Underscores attached to words represent a word that was italicized.

Contribution

Sean wrote the tokenizer for the raw text with both regular expression and NLTK. She wrote the initial unigram and bigram calculator that created a dictionary of key value pairs for the n-grams and total occurrences (version 1). Sean also wrote the project report.
Molly made adjustments to the unigram and bigram calculator to make it fit more closely with her updates (version 2). She also wrote code to read in all of the text files in the training directory and calculate the probability of unigrams and bigrams. Finally, she wrote the entirety of the word_gen.py, our sentence generator. She

created sentence generation for unigrams randomly, and sentence generators for bigrams based on probability.