Cleaning:

1. remove all punctuation, spaces, and numbers from training data string
2. remove punctuation missed by split and numbers from test string

Model creation:

1. Unigram models built by passing training data string into FreqDist constructor(automatically counts occurrences
2. Bigram models built using ConditionalProbDist which creates internal FreqDists for each condition(n-1 char). bigrams() library function gives bigrams which are passed in as list of tuples(cond,pred)
3. Trigram done the same way just using a tuple (n-2,n-1) as condition

Probability calculation:

1. For every n-gram in training data calculate frequency of n-gram (method in FreqDist)
2. Get product of frequencies for n-grams in test word for each language(for loop to multiply frequencies)

-if freq 0, use Wittem-Bell discounting

1. Compare probabilities of each language model for each word(choose language with higher probability)
2. All test words are English, so sum predictions for english then divide by N to get accuracy

Challenges:

* Understanding how to use ConditionalProbDist
  + found explanation on StackOverflow, NLTK has no API only source code
* iterating through a given test word’s constituent n-grams
  + solved using slices with bounds calculated with n value
* Cleaning train and test set
  + created function to generate a new string from all alphabetic chars
* Implementing Wittem-Bell smoothing
  + since only alphabetic chars included max number of types = 26 ^ n

Output:

Uni-model accuracy: 657 / 981 = 0.6697247706422018

Bi-model accuracy: 426 / 899 = 0.4738598442714127

Tri-model accuracy: 69 / 981 = 0.07033639143730887

Unigram model predicted language best, likely because it gets more tokens which means more data to train on.