In [ ]:

```
import pickle
import math
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
```

Each bigram's probability with Laplace smoothing is: (b + 1) / (u + v) where b is the bigram count, u is the unigram count of the first word in the bigram, and v is the total vocabulary size (add the lengths of the 3 unigram dictionaries).

In [ ]:

```
def unpickler(language):
    Unigram = pickle.load(open(f'{language}_Uni.pickle', 'rb')) # read binary
    Bigram = pickle.load(open(f'{language}_Bi.pickle', 'rb'))
    return Unigram, Bigram
```

In [ ]:

```
def calc_prob(each_line, unigram_dict, bigram_dict, V):
    test_unigrams = word_tokenize(each_line)
    test_bigrams = list(ngrams(test_unigrams, 2))

    p_laplace = 1 # laplace smoothing

    for each_bigram in test_bigrams:
        n = bigram_dict[each_bigram] if each_bigram in bigram_dict else 0
        d = unigram_dict[each_bigram[0]] if each_bigram[0] in unigram_dict else 0
        p_laplace = p_laplace * ((n + 1) / (d + V))

    return p_laplace
```

In [ ]:

```
def main():
    # Read in your pickled dictionaries
    E_Uni_dict, E_Bi_dict = unpickler('E')
    F_Uni_dict, F_Bi_dict = unpickler('F')
    I_Uni_dict, I_Bi_dict = unpickler('I')

    # read in test file
    with open('LangId.test', 'r') as f:
        test_file = f.readlines()
    f.close()

    # read in solutions file
    with open('LangId.sol', 'r') as f:
```

```
    solutions_file = f.readlines()
f.close()

#For each line in the test file, calculate a probability for each language
#and write the language with the highest probability to a file
V = len(E_Uni_dict) + len(F_Uni_dict) + len(I_Uni_dict)

for each_line in test_file:
    E_prob = calc_prob(each_line, E_Uni_dict, E_Bi_dict, V)
    F_prob = calc_prob(each_line, F_Uni_dict, F_Bi_dict, V)
    I_prob = calc_prob(each_line, I_Uni_dict, I_Bi_dict, V)

    #if english has highest probability
    if E_prob > F_prob and E_prob > I_prob:
        with open('highest_prob.txt', 'a') as f:
            f.write("English\n")
        f.close()

    #if french has highest probability
    elif F_prob > E_prob and F_prob > I_prob:
        with open('highest_prob.txt', 'a') as f:
            f.write("French\n")
        f.close()

    #if italian has highest probability
    else:
        with open('highest_prob.txt', 'a') as f:
            f.write("Italian\n")
        f.close()

# read in from predictions file
with open('highest_prob.txt', 'r') as f:
    highest_prob = f.readlines()
f.close()

# keep track of line numbers that are incorrectly classified
inc_line_num = []



#calculate accuracy of predictions
correct_prediction = 0

for each_line in range(1,len(solutions_file)):
    if highest_prob[each_line] == solutions_file[each_line]:
```

```
            correct_prediction += 1
        else:
            inc_line_num.append(each_line + 1)     # +1 because list index will start at line 0

    # Compute and output your accuracy as the percentage of correctly classified instances
    accuracy = correct_prediction / len(highest_prob)

    # output your accuracy, as well as the line numbers of the incorrectly classified items
    print("Accuracy: ", accuracy)
    print("Incorrect Classification line numbers: ", inc_line_num)
```

In [ ]:

```
if __name__ == "__main__":
    main()
```

In [ ]: