Note to the reader: the location of all files needs to be the same for all files, in a local directory.

Tasks:

3.3.1) Calculating the perplexity of the string [[abaab]

There are 8 characters in the sequence, but since we are working with trigrams, our count will be from 1 to 6.

[	[	A	b	a	a	b	]
-1	0	1	2	3	4	5	6

Calculating perplexity:

PP(W) = $(\prod_{i=1}^{n=6} P(W_i | W_{i-2}, W_{i-1}))^{-1/6}$ =

$= P(W_1 | W_{i-1}, W_{i0}) \times (W_2 | W_{i0}, W_{i1}) \times P(W_3 | W_{i1}, W_{i2}) \times P(W_4 | W_{i2}, W_{i3}) \times P(W_5 | W_{i3}, W_{i4}) \times P(W_6 | W_{i4}, W_{i5})^{-1/6}$

PP(W) = (P (a | [, [ ) x P (b | [, a) x P (a | a , b) x P (a | b , a) x P (b | a , a) x P ( ] | a , b)) -1/6 =

= (0.2 x 0.7 x 0.6 x 0.25 x 0.5 x 0.1) -1/6 = (0.00105) -1/6 ≈ **3.14**

We have also manually calculated the perplexity of a simpler test case

W = [ba] PP(W) ≈ **8.16**

Both functions have shown similar results when run through Python (3.1366 and 8.1649).
Code starting line 638.

3.3.2) Cleaning up the data:

Name of the function: preprocess_line

This functions pre-processes the data so that all non-alphabetic characters with the exception of “coma”, “space”, “period” are deleted, all numbers are converted to 0, all uppercase letters are converted to lowercase. It makes use or RegEx syntax. Code shown below:

```
def preprocess_line(text):
    if not text:
        print ("Empty string in preprocess() function. Exiting...")
        sys.exit(1)
    text1 =  re.sub('[^a-z^A-Z^0-9 ,.]','', text.lower())
    return re.sub('[\d]','0', text1)
```

3.3.3) Building the trigram character language model:

Name of the function: generate_lang_model_en/es/de()
There are three functions, for each individual language model.
They all call the function estimate_log_probabilities_with_add1()

Name of the function: estimate_log_probabilities_with_add1()

This function is used to collect the occurrences of the trigrams. Unseen trigrams are initially assigned a count of 0, and increased by 1 every time the same occurrence is found.

Data Structure

Nested dictionaries. These are referred to as outer dictionary and inner dictionary. The outer dictionary’s Keys contains the third character of the trigram. The outer dictionary’s value is the inner dictionary.
The inner dictionary’s Keys are the list of bigrams associated with the outer dictionary’s keys. The inner dictionary’s values are the counts of occurrences of trigrams, later used for the calculation of probabilities.

Data structure to store probabilities

Both the probability of the bigrams and trigrams are stored in two different default dictionary data structure.

Calculating probability and Smoothing method

We have implement a smoothing method to avoid the risk of having trigrams with 0-values and probability, considering that these unseen trigrams might appear in the test set.

The smoothing method implemented is the add-one smoothing:

$$\text{Add-one smoothing: } P^*(Wn|Wn_{-2}, Wn_{-1}) = \frac{C(Wn-2,\ Wn-1,\ Wn)+1}{C(Wn-2,\ Wn-1)+V}$$

Where *V* stands for the size of the vocabulary, which in this case is 30 characters. We chose this method because, 1) we know the size of the vocabulary, and 2) because V is quite small, so the inclusion of this number into the equation does not modify heavily the original probabilities of non-0 trigrams.

Function log_estimate_probabilities: All estimated probabilities are converted to a log value (to the base 2).

Function get_history_english("th")

Returns a new file with the extract of all trigrams with history *"th"*, and their probabilities. This only takes as a basis the feed from the English LM. Trigrams with probability 0.000259403372244 are the result of smoothing, so they don't really appear in the training file.

Trigram	Probability
th	0.0539559014267
th,	0.00181582360571
th.	0.00207522697795
th0	0.000259403372244
tha	0.125032425422
thc	0.000518806744488
thb	0.000259403372244
the	0.65551232166
thd	0.00129701686122
thg	0.000259403372244
thf	0.000259403372244
thi	0.120881971466
thh	0.000259403372244
thk	0.000259403372244
thj	0.000259403372244
thm	0.000259403372244
thl	0.000778210116732
tho	0.0184176394293
thn	0.000259403372244
thq	0.000259403372244
thp	0.000259403372244
ths	0.00285343709468

The 3 trigrams with highest probabilities are *"tha", "the", "thi"*. Having thought about the problem beforehand, we had anticipated that these 3 would bring high probability values, being the highest the value of *"the"*, for its role as an article in English. The other two *"tha"* and *"thi"* are highly used in "*that, thank, than"*, which for different reasons are very much used in English, and *"this, think, thing* (including the pronouns *"something", "everything", "anything"…*)".

Function get_history_english("an")

Returns a new file with the extract of all trigrams with history *"an"* and their probabilities. This only takes as a basis the feed from the English LM. Trigrams with probability 0.000576368876081 are the result of smoothing.

Trigram	Probability
an	0.171181556196
an,	0.00230547550432
an.	0.000576368876081
an0	0.000576368876081
ana	0.0167146974063
anc	0.0593659942363
anb	0.000576368876081
ane	0.00403458213256
and	0.478962536023
ang	0.028242074928
anf	0.000576368876081
ani	0.0149855907781
anh	0.000576368876081
ank	0.021325648415
anj	0.000576368876081

anm	0.000576368876081
anl	0.000576368876081
ano	0.00691642651297
ann	0.0276657060519
anq	0.000576368876081
anp	0.000576368876081
ans	0.0674351585014

The results of this trigram are very interesting. The highest probabilities are associated to the trigrams *"and", "an ", "ans", "anc"*. Having thought about the problem beforehand, we knew we would find in the list *"an "* and *"and"*, given their grammatical function of article and conjunction. It was interesting though to find *"anc"* and *"ans"*. Having scrolled through the English training file, the trigram "anc" occurs in words such as: *financial, importance, significance, circumstance, guidance, accordance, allowance…* The use of all these words reveals the type of vocabulary used throughout the document, which indeed is bureaucratic / administrative. On the other hand, the trigram *"ans"* is most commonly found in words such as: *transport, transportation, transported,* which reveal a recurrent topic in the document.

3.3.4) Generating random output

Method used: Shannon visualization method: starting from a trigram looking *<s><s>C*, with one of the highest probabilities. Based on C|C-2, C-1, we choose the following trigram with the highest probability and so on. The model looks like this:

<s><s>C_3
<s> C_3 C_4
C_3 C_4 C_5
C_4 C_5 C_6
C_5 C_6 C_7
………… C_{298} C_{299} </s>

In the actual code we could not implement the start of sentence as *<s><s>C,* so it is represented as *". C"*, which is period + space + Character with high frequency.

Random output, 300 characters, English:

. of the the the the the the the the the the the the the the the the the the the the the the the the the the the the
the the the the the the the the the the the the the the the the the the the the the the the the the the the the the
the the the the the the the the the the the the the the the the the th

Random output, 300 characters, German:

. der der der der der der der der der der der der der der der der der der der der der der der der der der der der
der der der der der der der der der der der der der der der der der der der der der der der der der der der der
der der der der der der der der der der der der der der der der der der der d

Random output, 300 characters, Spanish:

. kocin es de la cones de la cones de la cones de la cones de la cones de la cones de la cones de la cones de la
cones de la cones de la cones de la cones de la cones de la cones de la cones de la cones de la cones de la cones
de la cones de la cones de la cones de la cones de la cones de la cones de l

These results are very interesting and we have been investigating a lot before validating they are actually correct. The explanation is that the output given goes into a loop whenever it encounters the starting of the most probable trigrams. In fact, these are short words and at least for English and German they are 2 articles, *the* for English and *der* for German. For Spanish, the sequence goes through the verb "to be" *es*, the preposition "of" *de*, the article "the" *la* and a non-existing word formed by high-prob trigrams *cones*.

One of our references here was Jurafsky, https://class.coursera.org/nlp/lecture/14, min 06.38, where he states: "The word "the" is very frequent in English. A unigram language model does not depend on surrounding words, so "the the the the" gets a high probability even though it isn't regularly used". In our case, our trigram model neither depends on the surrounding words, so it just generates text according to the frequency of the trigram.

3.3.5) Reading a test document and calculating its perplexity under the estimated language models.

To test the results of the perplexity function, we have tested the sequence *[[abaab]* (Ref. task 3.3.1). Results show that the manual calculation and the calculation from the function return similar values.
The function used is the following one:

$$PP(W) = \left(\prod_{i=1}^{n} P(W_i | W_{i-2}, W_{i-1}) \right)^{-1/n}$$

The perplexity showing the lowest result is the one indicating the language the file is written in, considering that by lowering the perplexity, the probability of the LM is maximised.

Question 3.3.5 Following are the results of perplexities calculated for each of the three langugage models - english, spanish and german

The perplexity for english for test file is 9.08976036231
The perplexity for spanish for test file is 22.9409380865
The perplexity for german for test file is 23.3176775109
The detected language for test file is English

Additional manual tests performed, and their results:

The perplexity for english for text = the deepening of the internal market is 7.1384623176
The perplexity for spanish for text = the deepening of the internal market is 23.1344146114
The perplexity for german for text = the deepening of the internal market is 20.8628459402
The detected language for text = the deepening of the internal market is English

The perplexity for english for text = la comisin debe continuar avanzando por este is 12.8067678054
The perplexity for spanish for text = la comisin debe continuar avanzando por este is 6.08615335435
The perplexity for german for text = la comisin debe continuar avanzando por este is 21.7123468093
The detected language for text = la comisin debe continuar avanzando por este is Spanish

The perplexity for english for text = bestehende system der europischen wettbewerbsregel is 23.8737858172
The perplexity for spanish for text = bestehende system der europischen wettbewerbsregel is 29.3370194245
The perplexity for german for text = bestehende system der europischen wettbewerbsregel is 6.84768156088
The detected language for text = bestehende system der europischen wettbewerbsregel is German

Eventually both a unigram and bigram model could work as well, if some specific characters are widely used in one of the 3 languages, or a 2-character combination is widely used in a specific language.
Nevertheless, this has not been tested and cannot be proofed.

In the event that we are given a new test document and its perplexity under our English LM, we could predict if the document is written in English depending on: the length of the test document (if very short, we could not be sure of the significance of the perplexity calculated), and whether the perplexity is lower than the one calculated over the current test set (if so, there are probabilities that the new test set is in English as well). If not under these circumstances, we could not make any assumptions.

3.3.6) Extending our work

- For all functions, we have added edge cases so that the program does not return an error if non-valid input is provided. The program is tested and robust.
- We have taken into account the possibility of having unseen characters in the test set so it can be opened without throwing an error. We have taken into account the possibility of having unseen characters in the test set so it can be opened without throwing an error.

Additional Web References used:
For Language Modelling:
Standford University:

- https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf
- https://www.youtube.com/watch?v=s3kKlUBa3b0&list=PL6397E4B26D00A269&index=12

For Python syntax:
Stack Overflow:

- http://stackoverflow.com/questions/613183/sort-a-python-dictionary-by-value
- http://stackoverflow.com/questions/31723719/how-to-use-a-specific-data-structure-as-the-default-factory-for-a-defaultdict/31723862#31723862