ASSIGNEMENT 1 REPORT
NLU

Francesco Ferrini

Exercise 1:

Define a function to extract a path of dependency relations from the ROOT to a token

In the **getDependency** function a sentence is given as input. Then it is parsed getting a Doc object of spacy (**doc**).
The for loop iterates over all the tokens in the sentence. The condition in the while loop puts in a list (**list1**) all the tokens (**token.text**) and the dependencies (**token.dep_**) until the token is the token.head (it means until it reaches the root of the sentence).
At the end of the for loop the list1 is reversed and is appended to the dependency list (**dep_list**).
This is an example of input/output:

```
Input: “The fire was caused by the gas stove and not by the gas tank.”
Output:
['ROOT:', caused, '|', 'nsubjpass:', 'fire', '|', 'det:', 'The']
['ROOT:', caused, '|', 'nsubjpass:', 'fire']
['ROOT:', caused, '|', 'auxpass:', 'was']
['ROOT:', caused]
['ROOT:', caused, '|', 'agent:', 'by']
['ROOT:', caused, '|', 'agent:', 'by', '|', 'pobj:', 'stove', '|', 'det:', 'the']
['ROOT:', caused, '|', 'agent:', 'by', '|', 'pobj:', 'stove', '|', 'compound:', 'gas']
['ROOT:', caused, '|', 'agent:', 'by', '|', 'pobj:', 'stove']
['ROOT:', caused, '|', 'agent:', 'by', '|', 'cc:', 'and']
['ROOT:', caused, '|', 'agent:', 'by', '|', 'conj:', 'by', '|', 'neg:', 'not']
['ROOT:', caused, '|', 'agent:', 'by', '|', 'conj:', 'by']
['ROOT:', caused, '|', 'agent:', 'by', '|', 'conj:', 'by', '|', 'pobj:', 'tank', '|', 'det:', 'the']
['ROOT:', caused, '|', 'agent:', 'by', '|', 'conj:', 'by', '|', 'pobj:', 'tank', '|', 'compound:', 
'gas']
['ROOT:', caused, '|', 'agent:', 'by', '|', 'conj:', 'by', '|', 'pobj:', 'tank']
['ROOT:', caused, '|', 'punct:', '.']
```

Exercise 2:

Extract subtree of a dependents given a token

In the **subtreeExtraction** function a sentence is given as input. Then it is parsed getting a Doc object of spacy (**doc**).
A dictionary (**my_dict**) is created in order to be returned by the function.
The for loop iterates in the number of tokens in the sentence. For every token is inserted a new key in the dictionary and is found a subtree (**give_subtree**) for that particular token.
In the second for loop every token in the subtree, is inserted in the dictionary in the related list.
In this example the first key of the dictionary is “The” and the related list is empty because the subtree of his dependants is empty.

```
Input: “The fire was caused by the gas stove and not by the gas tank”
Output: {The: [], fire: ['The'], was: [], caused: ['The', 'fire', 'was', 'by', 'the', 'gas', 
'stove', 'and', 'not', 'by', 'the', 'gas', 'tank', '.'], by: ['the', 'gas', 'stove', 'and', 'not', 
'the', 'gas', 'tank'], the: [], gas: [], stove: ['the', 'gas'], and: [], not: [], by: ['not', 'the', 
'gas', 'tank'], the: [], gas: [], tank: ['the', 'gas'], .: []}
```

Exercise 3:

Check if a given list of tokens (segment of a sentence) forms a subtree

In the **checkSubtree** function a sentence and a list of words is given as input. The for loop iterates over all the tokens of the sentence and extracting the subtrees. Then the if checks weather the list of words given as input is one of the subtrees. If yes the function returns True, otherwise False.

```
Input: “The fire was caused by the gas stove and not by the gas tank.”
Output “The token list ['The', 'fire'] forms a subtree”
```

Exercise 4:

Identify head of a span given its token

In the **spanHead** function a span is given as input where a span is a slice of a sequence. Then in span_length the number of tokens are counted. The span is trasformed in a Doc object using **as_doc().** The for loop iterates in the **span_as_doc** object for every token until the token is the same as the head (that means that is the root of this object and so the head of the span)

```
Sentence: “The fire was caused by the gas stove and not by the gas tank.”
Span Input: “by Brownback Republican Senator of”
Output “by”
```

Exercise 5:

Extract sentence subject, direct object and indirect object spans

In the **extractSpanInfo** function a sentence is given as input. Firstly a dictionary with 3 keys is created. The 3 keys are strings: “subj”, “dobj”, “iobj”. For every key, is created a list. In the list corresponding to the key “subj” are inserted all the words that forms a subject span. In the list corresponding to the key “obj” are inserted all the words that form a direct object span and in the list corresponding to the key “iobj” are inserted all the words that form an indirect object span.

```
Input: “Carla bought an iphone and gave it to her boyfriend”
Output: {'subj': ['Carla'], 'dobj': ['iphone', 'it'], 'iobj': []}
```