1. The obvious top-down approach for sentence segmentation for English texts is using symbols like – [ . ? ! ;] to separate the sentences. We can use regular expressions to match these separators and segment the sentences using these regular expressions.

The regular expression split statement in python is **re.split('([!?."]+)',text)**

1. The **top-down approach may not always do the correct sentence segmentation**, for example consider several cases where the sentence separation symbols can be used for some words like ‘Mr.Rao is visiting USA’. Here the simple segmentation approach would fail to segment the sentence properly.

Counter examples:

Discourse1 : Mr.Rao is vising USA. He will arrive here today.

Output1 : ['Mr.', 'Rao is vising USA.', 'He will arrive here today.']

Discourse2 : Consider fig 2.3.1. It is in page 36.

Output2 : ['Consider fig 2.', '3.', '1.', 'It is in page 36.']

1. The NLTK punkt sentence tokenizer uses an **unsupervised learning** algorithm to learn the sentence boundaries from the corpus, where the simple-top down approach uses predefined set of symbols and regular expressions to segment sentences. The NLTK punkt tokenizer will return a list of lists where each inner list is a sentence that is segmented. Punkt tokenizer can be considered as bottom-up approach.
2. The implementation for both naïve and punkt tokenizer is written in sentenceSegmentation.py file
3. In naive case a simple regular expression is used to segment the sentence.
4. The NLTK punkt pre-trained sentence tokenizer is shown in the same file.

The cases where one method performs better are :

1. top-down performs better than punkt:

Discourse: 'Rao is vising USA.He will arrive here today.’

naive: ['Rao is vising USA.', 'He will arrive here today.']

punkt: ['Rao is vising USA.He will arrive here today.']

Here punkt tokenizer failed to see ‘.’ Between USA and He is a separator, it would have worked if there is a space after ‘.’ In the sentence, but the top-down approach segmented correctly in this case.

1. nltk-punkt perfoms better than top-down:

Discourse: 'Mr. Rao is vising USA. He will arrive here today.

naive: ['Mr.', 'Rao is vising USA.', 'He will arrive here today.']

punkt: ['Mr. Rao is vising USA.', 'He will arrive here today.']

Here naïve approach fails because it uses ‘.’ In Mr as a sentence separator.

1. The simplest word-tokenization approach in English is using symbols like-[ whitespace . , ! ? ] etc for the separating the tokens . We simply consider any symbol other than alphanumeric characters as token seperators.

The python statement with regular expression for simple word tokenization is:

**re.split('([!\?.;:\s,()"])', txt)**

1. The treebank tokenizer uses some rules like splitting standard contractions and using punctuation, single quotes, periods etc,. to separate word tokens. All these constitute the **top-down** knowledge.