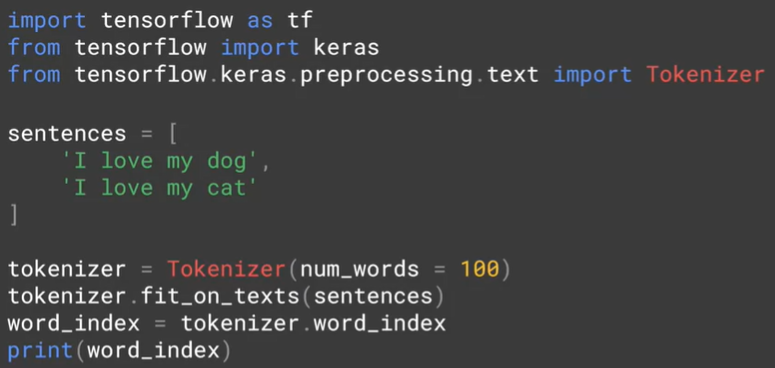
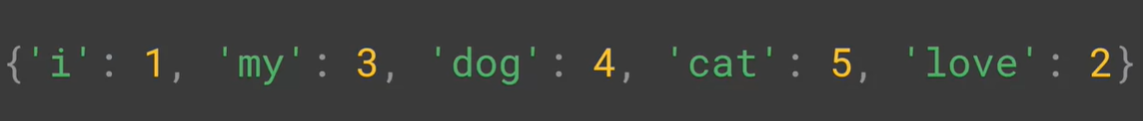
## From words to Numbers (words encoding):

* To pass from words to numbers (the classical way that our Neural Networks do the training and the prediction), we should give to each word a unique Id
* This process is included in the Tokenization process where we switch from String Data to vector of integers (the words index)

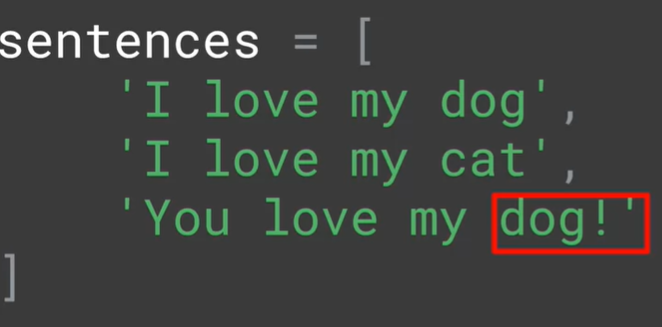
### The code :

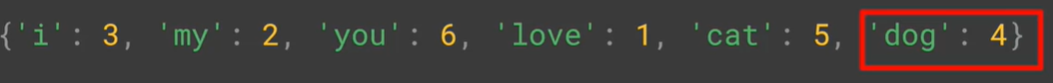


* We are going to use Tokenizer class from Keras
  + **Num\_words=100 :** 
    - it will take only the top 100 words which appears the most in our training text and encode them , for the other words : he will just ignore them
  + **Fit\_on\_texts(sentences) :**
    - Using this method , we will feed the Tokenizer by our text in order to encode its words
  + **.word\_index** :
    - ****It has the Python dict which contains our text words as Key and their corresponding number as Value:
    - The words appearing the most will have a lower Values

#### Interesting characteristics about the Tokenizer:

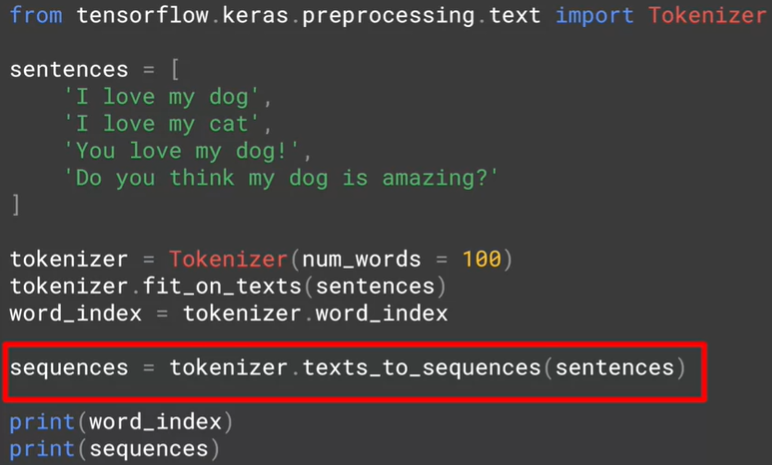
* In the image above, we see that it contains ‘i’ instead of ‘I’ and this because the Tokenizer lower all the word cases in order to normalize the text
* In the image below, we see by adding a new sentence with “dog!” as a word , Tokenizer is clever enough to consider it exactly as “dog” because it gets rid of the punctuation marks while doing the words encoding process

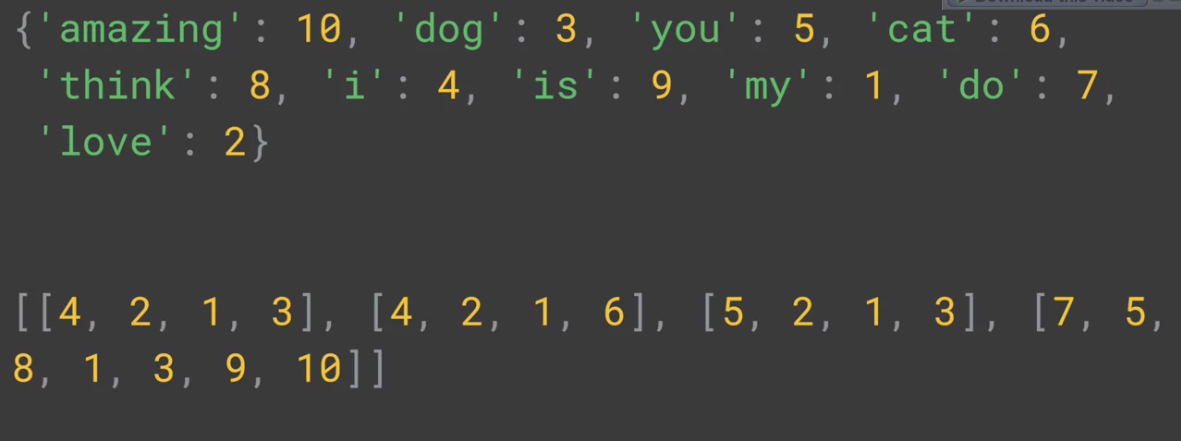


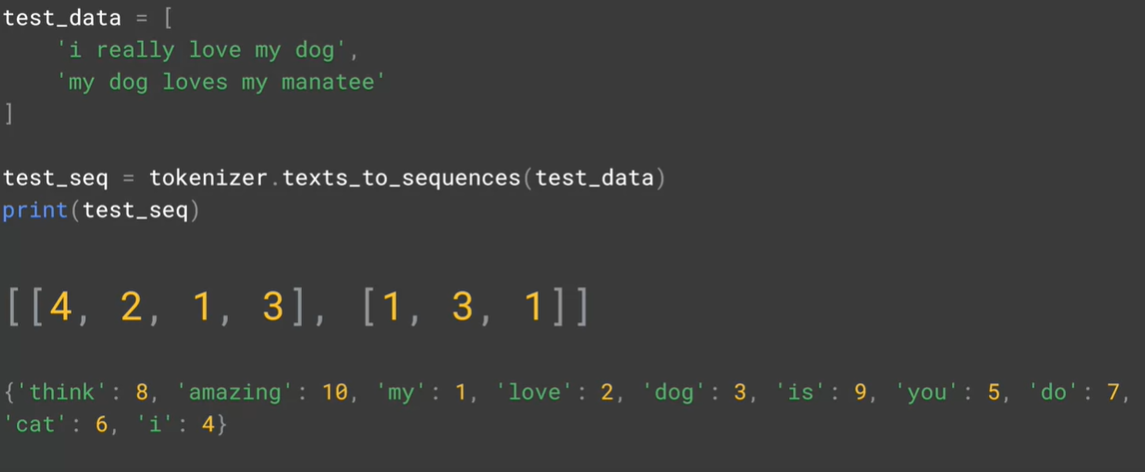


## Text to sequence + the Padding

### Convert my training data to tokens



* Using texts\_to\_sequence() method in my training data , this will convert my array of string to array of vectors of the corresponding tokens to each word.
* For the first row of the output : “I love my dog” was converted to [4,2,1,3] ( 4 = I , 2 = love , …etc )
* The tokens will be associated according to the last called method to .fit\_on\_texts()

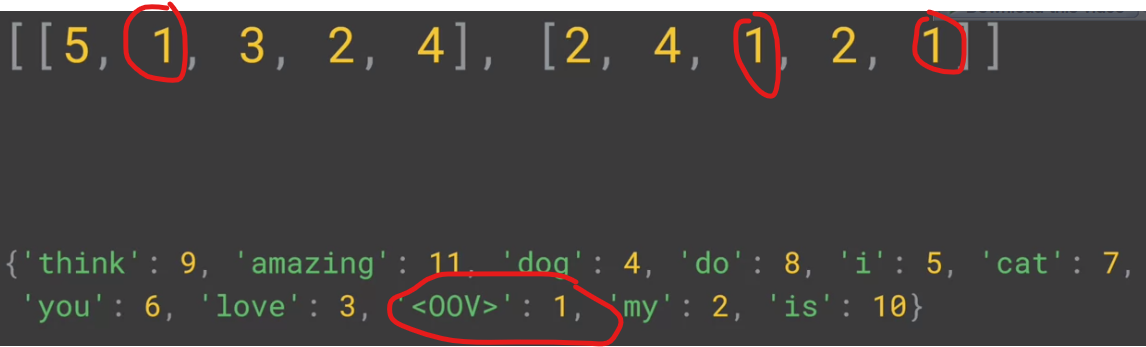


* In the image below , we tried to call texts\_to\_sequence() method in a new corpus which contains a new words like “really” and “manatee” comparing to the last time we called fit\_on\_texts() and “I really love my dog” gets the same sequence as “I love my dog”
* The conversion to sequence will simply ignore these unknown words to him

#### Making the tokenizer taking in consideration the unkown words :



* Thanks to the argument oov\_token , we are assigning the token ‘<OOV>’ to the unknown words compared to the fitting phase ( oov = out of vocabulary )
* The image below shows the final result after taking in consideration the unknown words:

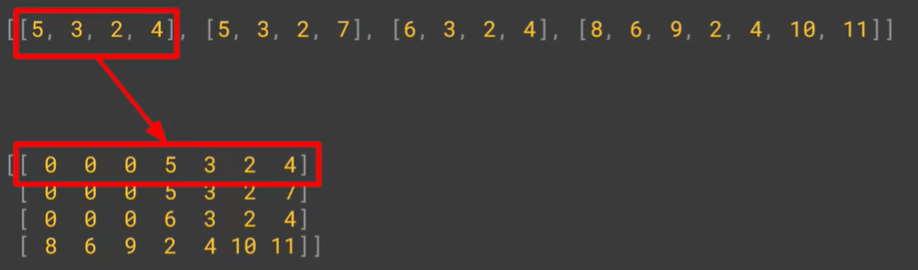


And now , the words “really” and “manatee” has 1 as a token

#### Padding the sequences:



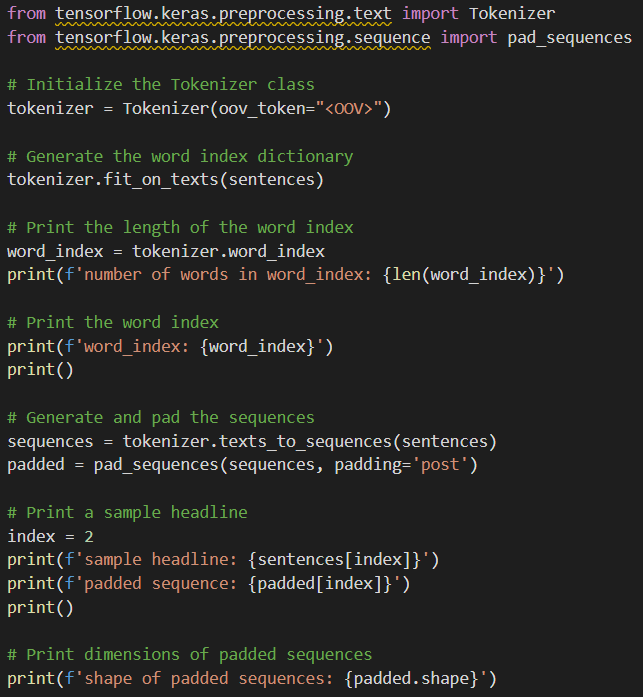
* you will usually need to pad the sequences into a uniform length because that is what your model expects : a constant length of sequence , and we will do that thanks to the method pad\_sequences() , it will add 0s in the left ( or the right ) of the sequences in order to make them in the same length ( usually equals to the longest sequence ) , and this is the final output :



* Interesting args to pad\_seqeunces :
  + padding=”post” / padding=”pre” :
    - It tells the method where it should put the 0s ( post = in the right and pre= in the left )
  + Maxlen= Number & truncating=’post’ / truncating=’pre’:
    - It tells the max size of the sequence after the padding process , it will cut words from the sentences who had longer length
    - The truncating argument tells the method where it should do the cut : from the right or the left

### Resming the pre-processing step :

1. Initialize the Tokenizer class
2. Calling fit\_on\_text() on the training corpus text :
   1. That corpus should be a huge text in order to cover the max possible words in the language in order to avoid the <OOV> token in our training set
   2. Its role is to generate a unqiue token to each word
   3. It returns Key-Value dict , the key field contains the words while the value contains their corresponding tokens
   4. We can check this dict by accessing to .word\_index attribute
3. Calling text\_to\_sequences() on our training set :
   1. Its goal is converting our array of sentences to array of sequences (vector of tokens)
4. Calling pad\_sequences() on the sequences array :
   1. Its goal is to unify the sentences length ( to be generally equal to the max length sentence by adding 0s in the post/pre sequence depending to truncating arg )



## Important Remark about max\_words argument in Tokenizer instantiation

* We know that the max\_words arg tells the Tokenizer class how many top words to take in consideration while tokenizing our sentences



* We see that the word\_index size is 29657 even though we specified max\_words=100
  + This because that the max\_words argument has nothing to do with the length of word\_index , in the all cases it will contain all the words seeing in the fitting
  + The difference is shown when we call texts\_to\_sequences() method :
    - If the word token in inferior than 100 : we will associate to him its token
    - If it’s superior we will associate to him the ‘<OOV>’ token
* The challenge with max\_words arg even it’s useful to eliminate the rare words : we have to make sure that the chosen tokens ( whose index inferior than max\_words ) must be semantic and useful words , and not just ‘to’ , ‘of’ , ‘the’ ..etc like we are seeing in the image above
  + One possible solution is to eliminate and delete these meaningless pronouns from the training set before calling fit\_on\_texts()