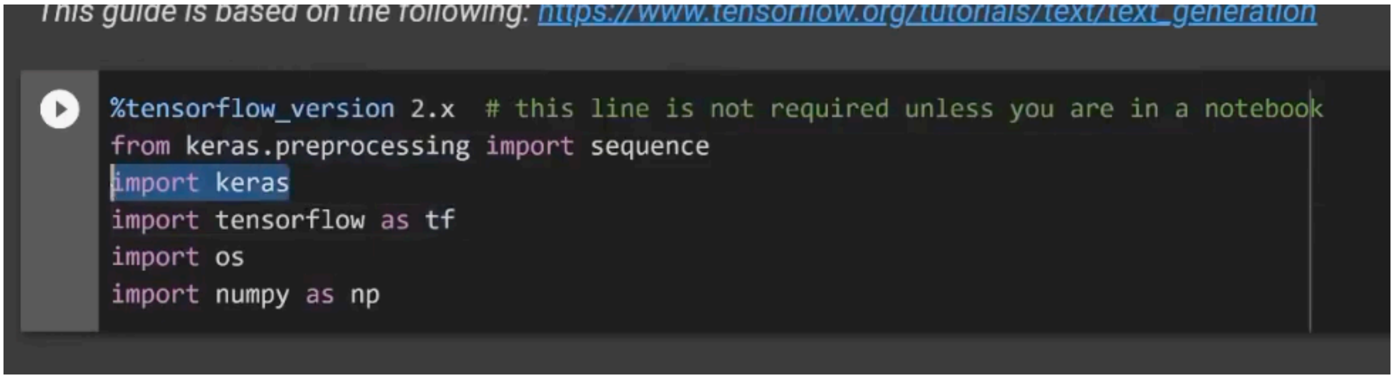
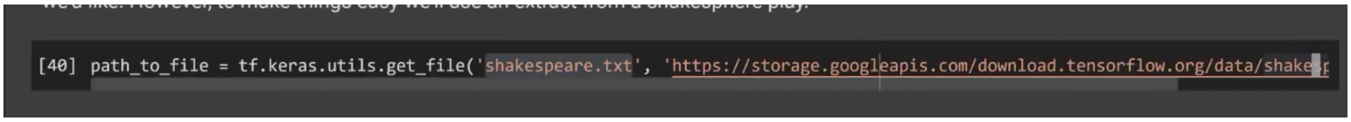
- ***creating a play generator which uses recurrent neural networks***
  - -> the input is a sequence - then it predicts the next word in the sequence
  - -> we are training it on passages from Romeo and Juliet -> it's going to predict a play
  - -> it predicts one letter at a time and because it's trained using a play this is what the model will predict
- ***importing the modules***



This guide is based on the following: https://www.tensorflow.org/tutorials/text/text_generation

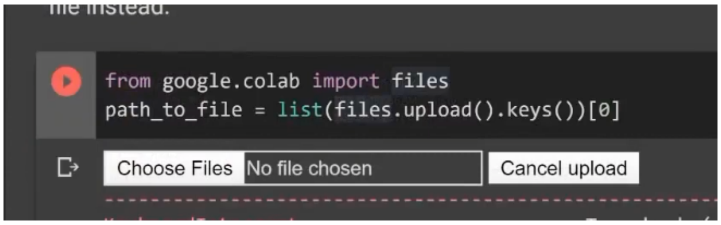
```
%tensorflow_version 2.x  # this line is not required unless you are in a notebook
from keras.preprocessing import sequence
import keras
import tensorflow as tf
import os
import numpy as np
```

  - -> he's imported keras, tensor flow, numpy and os
  - ***-> then importing the file with Romeo and Juliet***
    - -> this is the text to train the model, it imports the Romeo and Juliet files



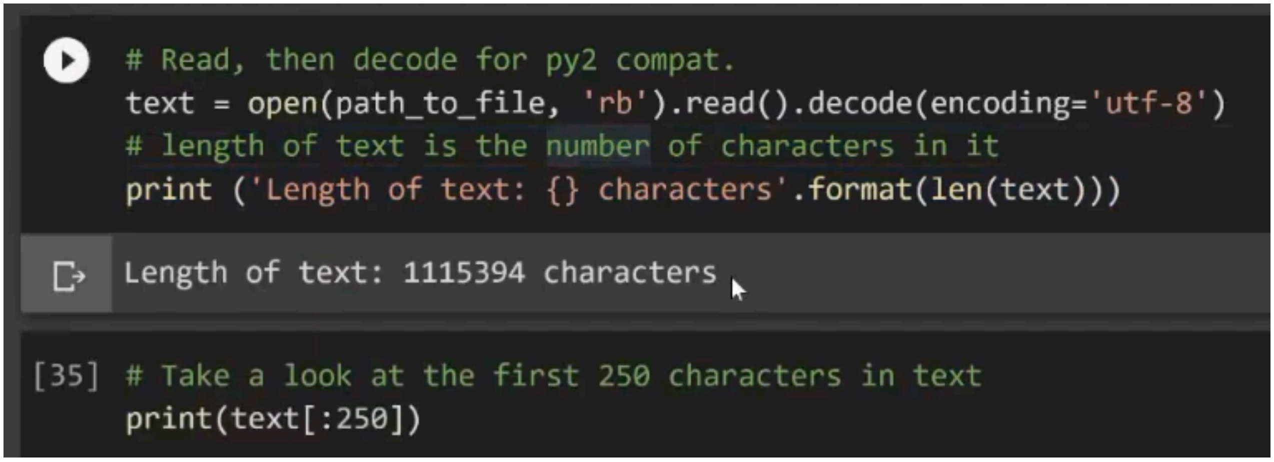
```
[40] path_to_file = tf.keras.utils.get_file('shakespeare.txt', 'https://storage.googleapis.com/download.tensorflow.org/data/shakesp
```

    - -> you can also do this using a txt file on your own machine



```
from google.colab import files
path_to_file = list(files.upload().keys())[0]
```

    - ***-> then to open the file with the training data***
      - -> in read mode (read bytes mode)
      - -> decoding it into utf-8 format
      - -> then printing out the length of it



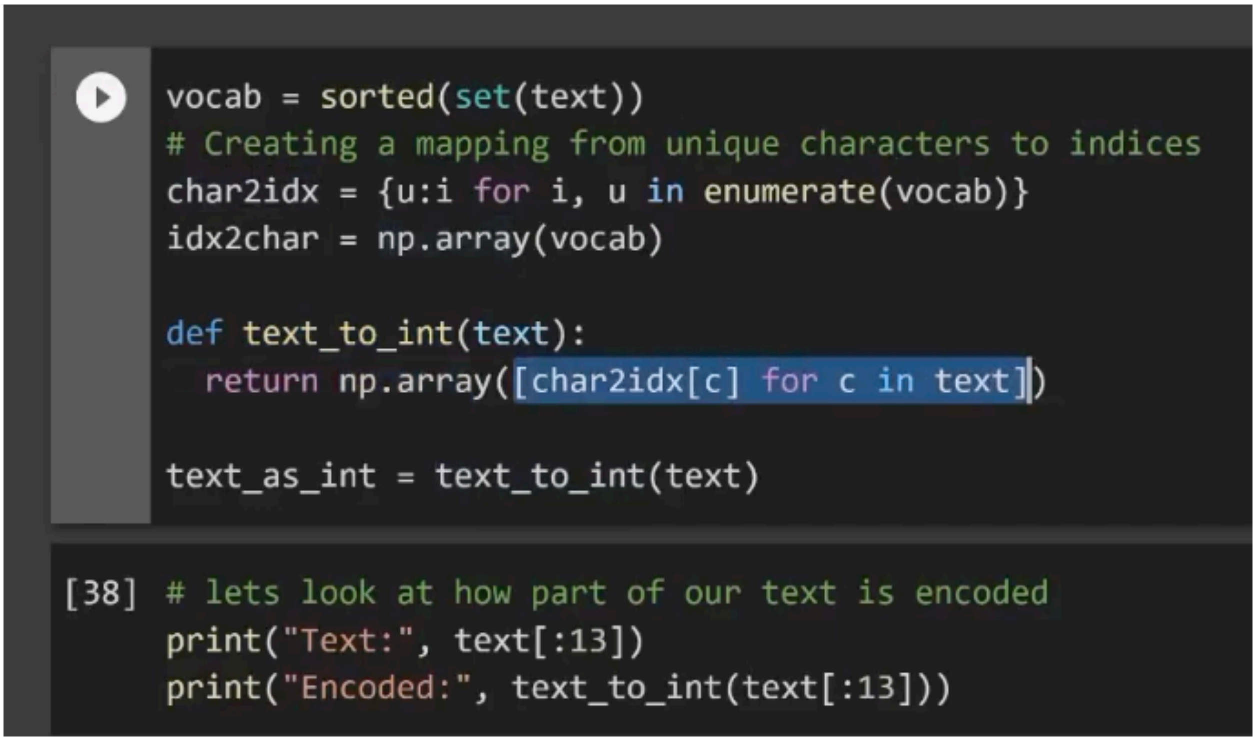
```
# Read, then decode for py2 compat.
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
# length of text is the number of characters in it
print ('Length of text: {} characters'.format(len(text)))
```

```
Length of text: 1115394 characters
```

```
[35] # Take a look at the first 250 characters in text
print(text[:250])
```

  - ***-> encoding***
    - -> we want to use Romeo and Juliet to train the model
    - -> we've just imported the play in
    - -> and now we want to change the text into integers - to use it to train the model
    - -> replacing each character in the text with an integer
    - -> there is an infinite amount of characters which could be encoded
    - -> encoding in a simple format

        - ***-> to do this***
            - -> sorting all of the unique characters in the text
            - -> creating a unique mapping
            - -> from unique characters to indices
            - -> we are giving every character in the play a character -> then converting that into an array
            - ***-> to go from letter to index and vice versa***



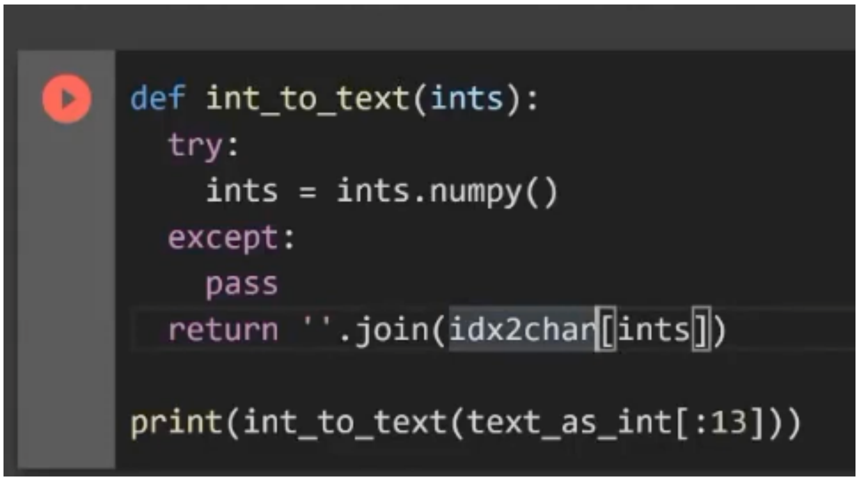
```
vocab = sorted(set(text))
# Creating a mapping from unique characters to indices
char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)

def text_to_int(text):
  return np.array([char2idx[c] for c in text])

text_as_int = text_to_int(text)
```

```
[38] # lets look at how part of our text is encoded
     print("Text:", text[:13])
     print("Encoded:", text_to_int(text[:13]))
```

            - ***-> figuring out how many unique characters are in the vocabulary***
                - -> sorting the elements in the text
                - -> turning the initial character into an array
                - -> taking text and converting it into an integer
                - -> doing that for the entire play
                - -> then putting it into a list
                - -> we've taken the entire play and converted the words into integers
                - -> each character also has its own number (e.g 'a' would be 1)
            - ***-> he's then defined a function which does this in reverse <- to be able to convert from the text in integer form to the text in word form***



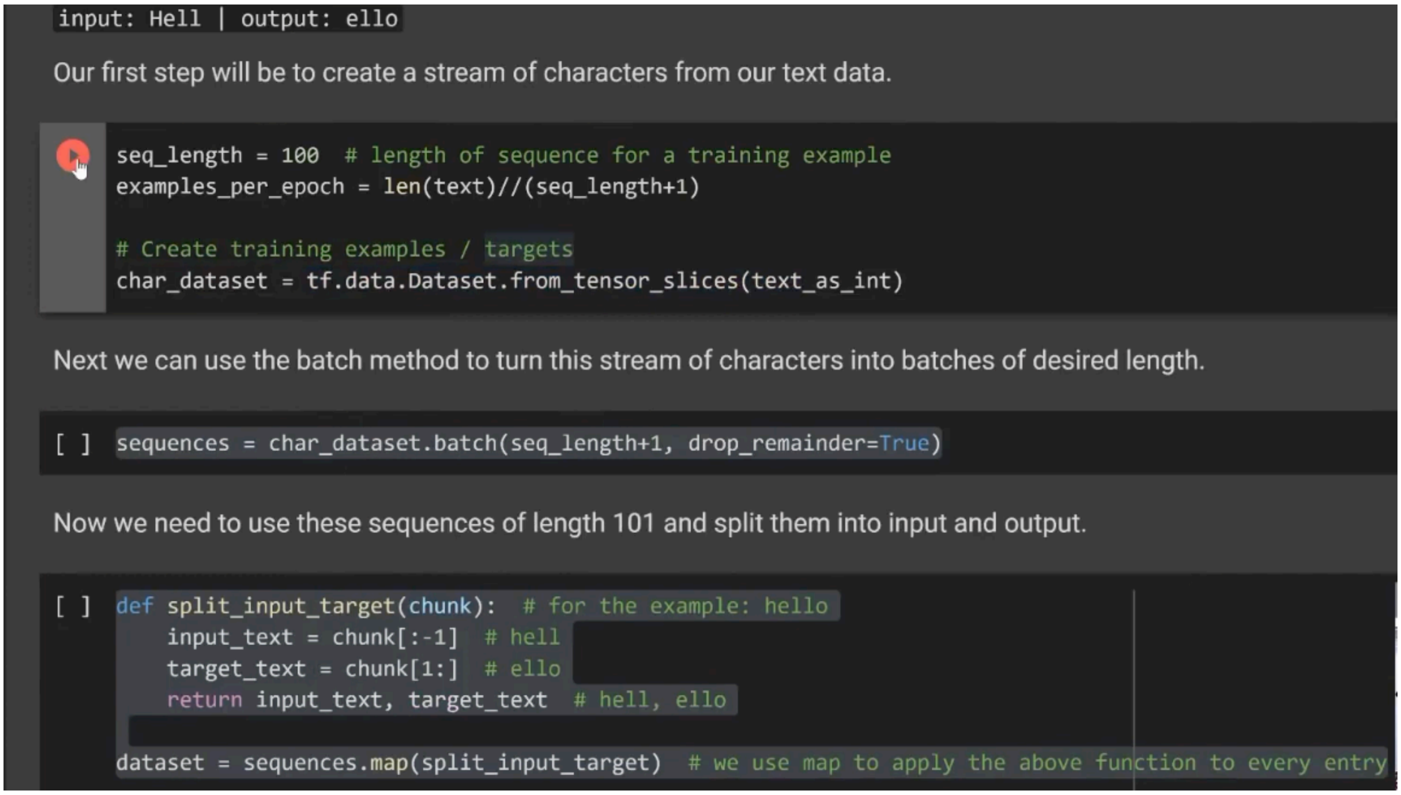
```
def int_to_text(ints):
  try:
    ints = ints.numpy()
  except:
    pass
  return ''.join(idx2char[ints])

print(int_to_text(text_as_int[:13]))
```

- ***splitting the play into different training examples***
    - -> different sections of text which the model can be trained on
    - -> we want different passages of the play which we can train the model on
    - -> the input is Hell and the output is ello <- the last character included and not the first (the

overall length of the word is the same)

- -> having 101 characters which you use for every training example
- -> converting the entire string dataset into characters



```
input: Hell | output: ello
```

Our first step will be to create a stream of characters from our text data.

```
seq_length = 100  # length of sequence for a training example
examples_per_epoch = len(text)//(seq_length+1)

# Create training examples / targets
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
```

Next we can use the batch method to turn this stream of characters into batches of desired length.

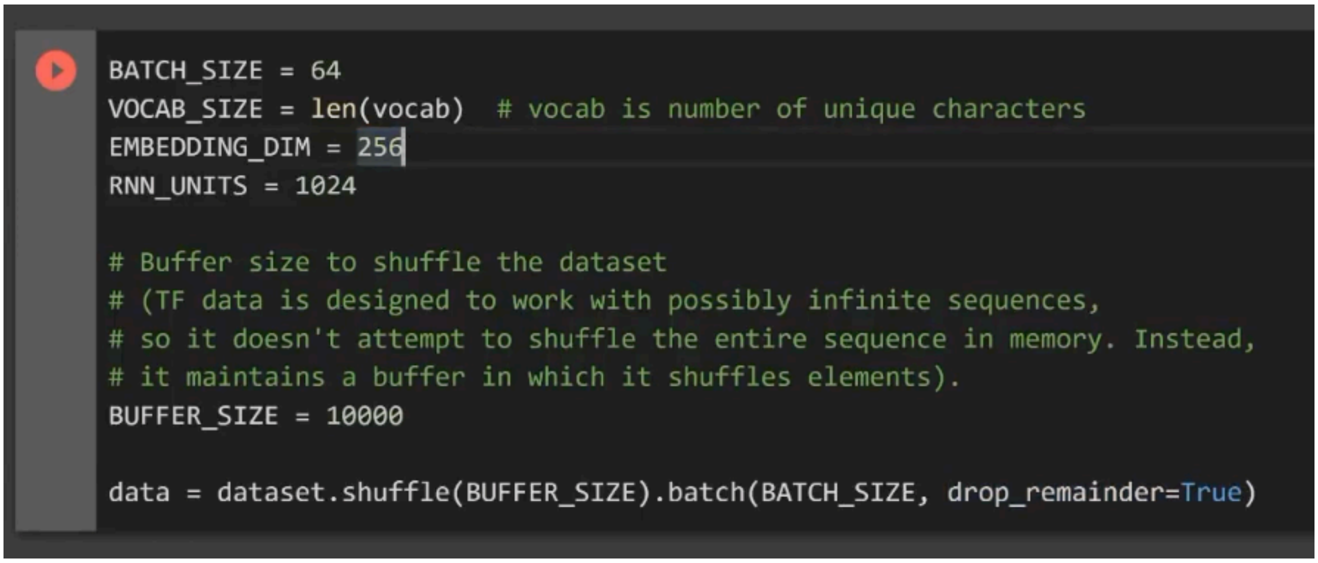
```
[ ] sequences = char_dataset.batch(seq_length+1, drop_remainder=True)
```

Now we need to use these sequences of length 101 and split them into input and output.

```
[ ] def split_input_target(chunk):  # for the example: hello
        input_text = chunk[:-1]  # hell
        target_text = chunk[1:]  # ello
        return input_text, target_text  # hell, ello

    dataset = sequences.map(split_input_target)  # we use map to apply the above function to every entry
```

  - ***-> the first cell is setting the length of the section of play which will be used for training the model***
    - -> then the number of training examples we want
    - -> asking it to create a sequence input and output
  - ***-> the next cell converts the entire string dataset into characters***
    - -> sequence length is the length of each batch (section of play which is being used to test the model)
    - -> drop remainder gets rid of the text which is above 100 characters
  - ***-> the third cell***
    - -> split input target <- this creates the training examples
    - -> converting the sequences into the input and target text
    - -> it's mapping it to another function
    - -> this is designed in a function
    - -> the results are stored in a dataset object
    - -> it's removing the first character from a string and adding a space on the end of it in the output
      - -> and it's all being done on the play which is being used to train the model
- ***-> then making the training batches***
  - -> setting the batch size
  - -> then the sorted set of the text <- the number of characters
  - -> then the units for the residual neural network
  - -> batching these training sets into 64 batches of data, after shuffling it
  - -> the embedding dimension is how big we want the vectors which represent the meanings of the word in the embedding dimension to be



```
BATCH_SIZE = 64
VOCAB_SIZE = len(vocab)  # vocab is number of unique characters
EMBEDDING_DIM = 256
RNN_UNITS = 1024

# Buffer size to shuffle the dataset
# (TF data is designed to work with possibly infinite sequences,
# so it doesn't attempt to shuffle the entire sequence in memory. Instead,
# it maintains a buffer in which it shuffles elements).
BUFFER_SIZE = 10000

data = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)
```

- -> then the RNN units