TASK 1.1 [2.5 marks]

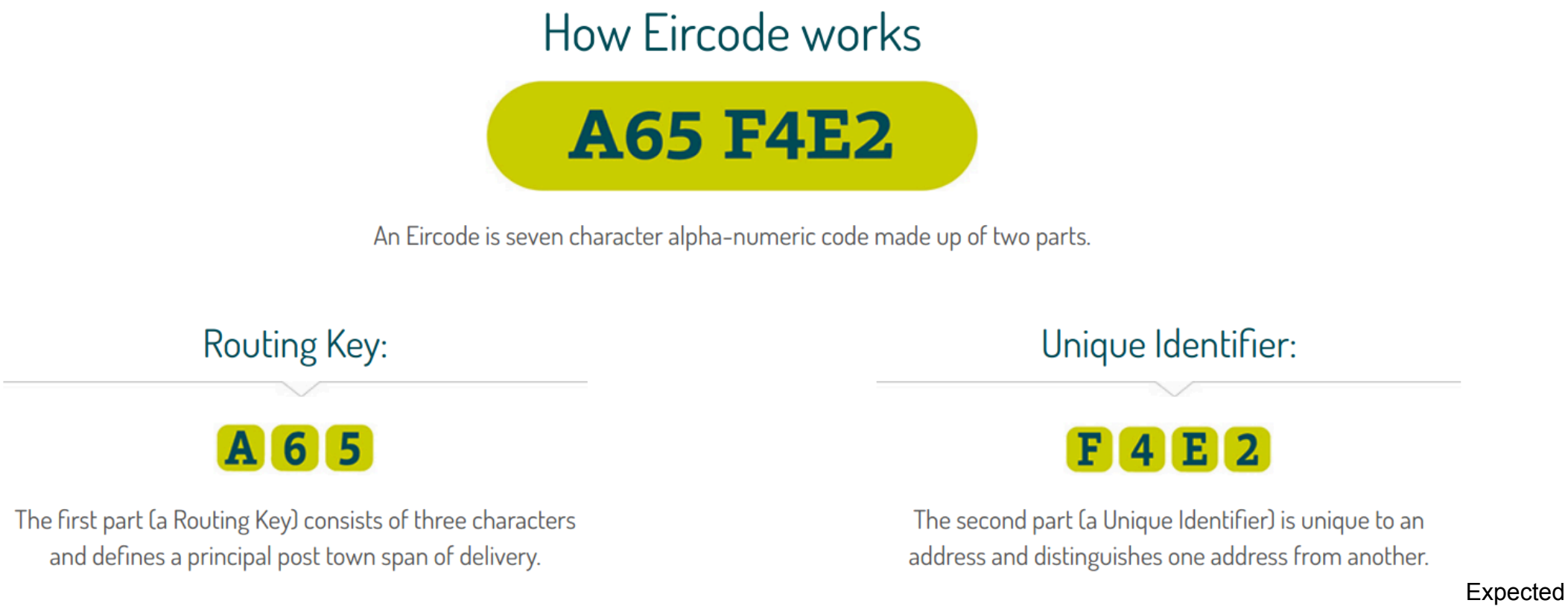
Write a function in Python that takes a string as input and uses regular expressions to:

A. check if the given string is a valid Eircode or not

B. if the string is a valid Eircode, identify and print out the Eircode's geographical district

- Each Eircode consists of seven CAPITAL letters and/or digits, in the format A65B2CD. The first three characters together represent the "Routing Key" part of the code and correspond to one of 139 unique geographical districts. A dash or space could (optional) separate the routing key from the unique identifier (i.e., the last 4 characters).
- You can download the full list of routing keys in CSV format from here using the !wget command.
- Use the 're' library for regular expressions in Python.

Regex101 is a handy website for trying and testing your regular expressions.



Expected Output:

- **Your eircodeValidator function should be able to handle all the supplied test cases.**

```
import re   # import re library for regex
import csv  # import csv library to handle the Eircode CSV file

# use the linux command wget to donwload the CSV file
!wget
https://gist.githubusercontent.com/ajoorabchi/eac194a79dd26de8864f9206b7842ff1/raw/8ea1d8d5f74b5b2724e378b43d4df6094990c7db/Eircode
%2520Routing%2520Key%2520Boundaries.csv
filePath = "/content/Eircode Routing Key Boundaries.csv" # set the path for the donwloaded CSV file

with open(filePath, 'r') as f:
```

```
    reader = csv.reader(f)
    KeyRouteList = list(map(tuple, reader)) #the map function iterates through the rows in the CSV file and puts them in tuples.
The tuples are then added to the KeyRouteList    [(k0,d0), ...,(kx,dx)]
print(KeyRouteList)

def eircodeValidator(eircode):
YOUR CODE HERE

eircodeValidator("111-T9PX")
eircodeValidator("V94-T9PX")
eircodeValidator("V94 T9PX")
eircodeValidator("V94T9PX")
eircodeValidator("   V94-T9PX")
eircodeValidator("V94-T9PX   ")
eircodeValidator("v94 T9PX")
eircodeValidator("V94T9PXV")
```

```
[('ROUTING KEY', 'DESCRIPTOR'), ('A41', 'BALLYBOUGHAL'), ('A42',

 Eircode = 111-T9PX
Valid Eircode pattern, Routing Key= 111, Unique Identifier= T9PX
111 is a valid but unassigned routing key
------------------------------------------------------------------

 Eircode = V94-T9PX
Valid Eircode pattern, Routing Key= V94, Unique Identifier= T9PX
Destination = LIMERICK
------------------------------------------------------------------

 Eircode = V94 T9PX
Valid Eircode pattern, Routing Key= V94, Unique Identifier= T9PX
Destination = LIMERICK
------------------------------------------------------------------

 Eircode = V94T9PX
Valid Eircode pattern, Routing Key= V94, Unique Identifier= T9PX
Destination = LIMERICK
------------------------------------------------------------------

 Eircode = V94-T9PX
Valid Eircode pattern, Routing Key= V94, Unique Identifier= T9PX
Destination = LIMERICK
------------------------------------------------------------------

 Eircode = V94-T9PX
Valid Eircode pattern, Routing Key= V94, Unique Identifier= T9PX
Destination = LIMERICK
------------------------------------------------------------------

 Eircode = v94 T9PX
Invalid Eircode pattern
```

Task 1.2 [2.5 marks]

Complete the **contactsExtractor** function in the code snippet below. The function should try to extract as many contacts (**phone numbers, emails, and websites**) from the contact information page of the UL website as possible.

Hint1: the re.**findall()** method iterates over a string to find a subset of characters that match a specified pattern. It will return a list of every pattern match that occurs in a given string.

```
import re   # import re library for regex
!pip install html2text
import html2text # import html2text library to convert and extract the text content of the HTML file

!wget https://www.ul.ie/contact-information # Use the Linux command wget to download the webpage
filePath = "/content/contact-information" # Set the path for the downloaded HTML file
contact_information_file = open(filePath, "r")
contact_information_html = contact_information_file.read()
contact_information_text = html2text.html2text(contact_information_html)
#print(contact_information_text) #uncomment to see the text content of the page

def contactsExtractor(contact_information_text):
YOUR CODE HERE

contactsExtractor(contact_information_text)
```

Expected Output:

n Phone Numbers found: ['0035361202700', …]

n E-mails found: ['reception@ul.ie', …]

n Websites found: ['www.ul.ie/library', …]

, where ***n*** = the total number of matches found.

- **Try to reduce the number of Type I (false-positive) and Type II (false-negative) errors as much as possible.**
- **Evaluate the performance of your Information Extraction function in terms of the number of Type I and Type II errors.**

Task 2.1

Use the *!wget* command to download the Complete Works of William Shakespeare from here; then open the downloaded text file and print out its first 50 lines.

Python String splitlines() Method Split a string into a list where each line is a list item.

```
#TASK 2.1

!wget http://www.gutenberg.org/files/100/100-0.txt

filePath = "/content/100-0.txt"

ShakespeareFile = open(filePath, "r")

ShakespeareContent = ShakespeareFile.read()

ShakespeareContent = ShakespeareContent.splitlines() # The splitlines() method splits a string into a list. The splitting is done at line breaks.

print(ShakespeareContent[0:50])
```

Task 2.2 **[2 marks]**

Use the tf.keras.preprocessing.text.Tokenizer class to:

a. Tokenize the Shakespeare corpus
b. Print out the total number of Tokens in the corpus
c. print out the total number of Types in the corpus

d. Print out the top 10 most frequent Types in the corpus along with their ranks and frequencies. For example:

```
"the" is ranked #1, with a frequency of xxxxx
.
.
.
```

Tokenizer

```
keras.preprocessing.text.Tokenizer(num_words=None, filters=base_filter(),
    lower=True, split=" ")
```

Class for vectorizing texts, or/and turning texts into sequences (=list of word indexes, where the word of rank i in the dataset (starting at 1) has index i).

- **num_words**: None or int. Maximum number of words to work with (if set, tokenization will be restricted to the top num_words most common words in the dataset).
- **Methods**:
  - **fit_on_texts(texts)**:
    - **Arguments**:
      - **texts**: list of texts to train on.
- **Attributes**:
  - **word_counts**: dictionary mapping words (str) to the number of times they appeared on during fit. Only set after fit_on_texts was called.
  - **word_docs**: dictionary mapping words (str) to the number of documents/texts they appeared on during fit. Only set after fit_on_texts was called.
  - **word_index**: dictionary mapping words (str) to their rank/index (int). Only set after fit_on_texts was called.
  - **document_count**: int. Number of documents (texts/sequences) the tokenizer was trained on. Only set after fit_on_texts or fit_on_sequences was called.

Task 2.3 **[2 marks]**

Use the nltk.stem.porter.PorterStemmer and nltk.stem.WordNetLemmatizer classes to:

a. stem all the Types in the Shakespeare corpus, and print out the total number of Types after stemming.
b. Lemmatize all the Types in the Shakespeare corpus, and print out the total number of Types in the corpus after lemmatization.
c. Confirm the validity of this arithmetic expression:

total_number_of_types > total_number_of_lemmatized_types > total_number_of_stemmed_types

```
#TASK 2.3

from nltk.stem import PorterStemmer

import nltk

nltk.download('wordnet')

from nltk.stem.wordnet import WordNetLemmatizer

ps = PorterStemmer()

lemmatizer = WordNetLemmatizer()

YOUR CODE HERE
```

Task 2.4 [1 mark]

Use the Sentence Segmentation module in the spaCy package to:

a. Segment the last 100 lines of the Shakespeare corpus into sentences
b. print out the segmented sentences and their total number.

HINT: 100 lines in a text file is just that, 100 lines; think of it as 100 lines on a sheet of paper; how many sentences are in that sheet? No way to know until you read and count. One sentence could occupy just half a line, whereas another sentence could occupy multiple lines. lines on a sheet of paper are physical concepts, whereas sentences are grammatical concepts. distinguishing the difference between the two is the goal of this exercise.

</END OF E-TIVITY>

Resources for Data Science and Statistical Learning students:

- A gentle introduction to OOP in Python (classes, methods, attributes, objects)

Teaching team

- Addition of BPE
- Level of difficulty (easy/fair/hard)
- TF tokenizer vs.NLTK tokenizer
- String Fundamentals, Concatenation, Indexing and Slicing: Python Basics
- Text to Word cloud
- Demo Zipf's law in the Shakespeare corpus - plot 1st 100 top freq words.
- 1.2 and 1.5 as main vs. complementary