Actividad 7 - Análisis de frecuencia de palabras

UNSAM Programa - Adaptado de Exactas Programa

Verano 2020

Hoy trabajaremos con libros reales y sus palabras.

1. Ejercicios con palabras y libros

Recordamos algunos comandos:
Para abrir un archivo de texto y leer línea por línea:

```
filename="archivo.txt"
fp = open(filename, encoding="utf-8")
for linea in fp:
    print(linea)
```

Una frase se puede separ en una lista de palabras usando el método `split()`:

```
>>> "esta, es una frase de varias palabras.".split()
['esta,', 'es', 'una', 'frase', 'de', 'varias', 'palabras.']
```

Una cadena se puede convertir a minúsculas usando el mértodo `lower()`:

```
 >>> "Expresion CON mayusculas.".lower()
 'expresion con mayusculas.'
```

El método `palabra.strip(chars)` le saca (del comienzo y del fin) a la `palabra` los signos que estén en la cadena `chars`:

```
>>>"... este es un ejemplo, solo, las comas del final se van,.".strip(' ,.')
"este es un ejemplo, solo, las comas del final se van"
```

Típicamente `strip` se usa para sacar signos de puntuación y de espacios (que incluyen signos de fin de línea) definiendo `chars = string.punctuation + string.whitespace` (¿faltan signos de puntuación castellanos?). Por último, `line = line.replace('-', ' ')` reemplaza todas las apariciones del guión por espacios en una línea.

1. Escriba una función llamada `tiene(palabra, letra)` que devuelva True si la palabra tiene la letra pasada como parámetro.
2. Escriba un programa que lea línea por línea el archivo `palabras.txt` (una larga lista de palabras en castellano) e imprima aquellas líneas (en este caso serán palabras) de más de 22 caracteres de largo.
3. Escriba un programa que lea el archivo `palabras.txt`, limpie las palabras de signos de fin de línea y las guarde en una lista llamada `palabras`.
4. Calcule y compare el procentaje de palabras en el archivo `palabras.txt` que contienen la vocal `a`, la vocal `e`, etc. (ojo con los acentos!).
5. Escriba un programa que lea las primeras $N = 100$ líneas de un archivo de texto, línea por línea, parta cada línea en palabras limpiando espacios y signos de puntuación, convierta estas palabras en minúscula y las imprima. En la carpeta encontrará algunos archivos `txt` que puede usar como ejemplo. O puede bajarse su libro favorito (en formato plano: txt) del proyecto Gutenberg, limpiarle el encabezado con un editor de textos y usarlo como ejemplo.

2. Ejercicios con diccionarios

Un diccionario es una entidad que relaciona entradas (llamadas claves o *keys*) con valores. Para crear un diccionario se usan las llaves ('{' y '}') en lugar de los corchetes ('[' y ']') que se usan para crear listas. Los corchetes en diccionarios se usan para referirse a una clave en particular, como se ve a continuación:

```
>>> mi_dict= {}                 #crea un diccionario vacio
>>> mi_dict['one']=1            #asocia a la clave 'one' el valor 1
>>> mi_dict['two']=2            #asocia a la clave 'two' el valor 2
>>> mi_dict['three']=3          #etc
>>> mi_dict['two']              #muestra el valor asociado a la clave 'two'
2
```

En la última línea arriba consultamos el valor de la clave 'two'. Da 2 tal como fue definido anteriormente. A continuación imprimimos el valor actual del diccionario:

```
>>> mi_dict                     #muestra estado del diccionario
{'one': 1, 'two': 2, 'three': 3}
```

También es posible modificar el valor de una clave

```
>>> mi_dict['two']+=100         #incrementa el valor de 'two' en 100
>>> mi_dict
{'one': 1, 'two': 102, 'three': 3}
```

Usaremos también el método `mi_dict.keys()` que me da todas las llaves que tiene definido un diccionario. Si quiere saber si una llave ya fue definida en un diccionario puede usar la expresión booleana "`llave in mi_dict`" que devuelve True o False.
Utilizaremos los diccionarios para contar cuántas veces aparece una palabra en un libro.

1. Lea su archivo favorito palabra por palabra como antes y use un diccionario para guardar la cantidad de veces que aparece cada una. Imprima las palabras que aprecen más de 20 o 30 o 100 veces (ajuste para que no sean demasidas). Repita imprimiendo solo aquellas de cinco caracteres o más.

2. (+) Opcional: Imprima las 20 palabras más utilizadas en el texto.

3. Escriba un programa que lea su archivo favorito palabra por palabra e imprima aquellas palabras que no están en la lista llamada `palabras` creada anteriormente. ¿De qué tipo de palabras se trata? ¿Puede reducir esta lista filtrando algunas palabras? (por ejemplo: sacar los números, o aquellas palabras que incluyen algún signo no previsto).

4. (++) Opcional[1]: Le asignaremos un ranking a las palabras, ordenadas por su frecuencia de aparición. La más frecuente tendrá rango 1, la siguiente 2, etc. Escriba un programa que lea un texto, calcule la frecuencia de cada palabra y las ordene en orden decreciente de frecuencia. A partir de esto, genere dos listas ordendas: una de frecuencias y otra de rangos de las palabras del texto.

5. (+++) Opcional: La ley de Zipf describe la relación entre rangos y frecuencias en lenguajes naturales (`https://es.wikipedia.org/wiki/Ley_de_Zipf`). Específicamente, esta ley empírica dice que la frecuencia f de aparición de una palabra de rango r es $f = cr^{-s}$ donde s y c son parámetros que dependen del lenguaje y del texto. Tomando logaritmo de ambos lados de la ecuación obtiene:

   $$\log(f) = \log(c) - s\log(r)$$

   Por lo tanto, si plotea $\log(f)$ versus $\log(r)$, debería obtener (aproximadamente) una recta con pendiente $-s$ y ordenada al origen $\log(c)$.

   Tomando logaritmos a las listas del ejercicio anterior obtendrá las listas `log_f` y `log_r`, que contienen respectivamente el logaritmo de las frecuencias y el logaritmo de los rangos. Grafique estos valores con `plt.plot(log_r, log_f)` y verifique si parecen una línea recta. ¿Puede estimar el valor de s?

   Compare las curvas obtenidas para diferentes textos.

[1] Busque en la web *"sort dictionary by values"*.