Clase 3 - IO

March 25, 2017

1 Introducción

En esta tercera clase hablamos sobre cómo leer y escribir archivos usando python. Pero primero realizaremos los ejercicios de la segunda clase.

2 Desarrollo de los ejercicios de la clase anterior

2.1 Primer ejercicio

```
In [1]: L1 = [i ** 2 for i in range(0,101)]
        L2 = [letra for letra in 'Miguel']

In [2]: print(L1)
        print(L2)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 3
['M', 'i', 'g', 'u', 'e', 'l']
```

2.2 Segundo ejercicio

```
In [3]: def contador(string):
            diccionario_de_letras = dict()
            for letra in string:
                if letra in diccionario_de_letras:
                    diccionario_de_letras[letra] += 1
                else:
                    diccionario_de_letras[letra] = 1
            return diccionario_de_letras

In [4]: contador('aaaababbA')

Out[4]: {'A': 1, 'a': 5, 'b': 3}
```

2.3 Tercer ejercicio

```
In [5]: def contadorpalabras(string):
            #Primero, limpiamos el string de acuerdo a las indicaciones.

            for conector in '(),.;:?¡¿![]{}-':
                string = string.replace(conector, '')
            string = string.lower()

            # Después, separamos las palabras usando .split
            listadepalabras = string.split(' ')

            # Por último, contamos
            diccionario_de_palabras = dict()
            for palabra in listadepalabras:
                if palabra in diccionario_de_palabras:
                    diccionario_de_palabras[palabra] += 1
                else:
                    diccionario_de_palabras[palabra] = 1
            return diccionario_de_palabras

In [6]: contadorpalabras('Hola, mi nombre es Miguel (y Miguel es también el nombre

Out[6]: {'colega': 1,
         'de': 1,
         'el': 1,
         'es': 2,
         'hola': 1,
         'mi': 1,
         'miguel': 2,
         'nombre': 2,
         'también': 1,
         'un': 1,
         'y': 1}
```

3 Algo más sobre imprimir strings

Es muy común que a la hora de escribir archivos o sacar resultados necesitemos manipular un string de tal forma que varíe dependiendo de una variable. Por ejemplo, supongamos que estamos sacando unas gráficas y que necesitamos cambiar su nombre de 'fig1' a 'fig2' a 'fig3' automáticamente. Para hacerlo, aprovechamos el método `format` para strings:

```
In [7]: for k in range(10):
            print('El número actual es {}'.format(k))

El número actual es 0
El número actual es 1
El número actual es 2
```

```
El número actual es 3
El número actual es 4
El número actual es 5
El número actual es 6
El número actual es 7
El número actual es 8
El número actual es 9
```

```
In [8]: lista = ['a', 'b', 'c', 'd']
        for elemento in lista:
            print('El elemento actual es {}'.format(elemento))
```

```
El elemento actual es a
El elemento actual es b
El elemento actual es c
El elemento actual es d
```

Es probable que necesitemos ir variando más de una variable, para eso adoptamos la siguiente notación:

```
In [9]: diccionario_de_edad = {'Miguel': 22, 'Diana': 49, 'Sara': 18}
        for persona in diccionario_de_edad:
            print('{llave} tiene {valor} años'.format(llave = persona, valor=diccio
```

```
Miguel tiene 22 años
Sara tiene 18 años
Diana tiene 49 años
```

Podemos agregarle más condiciones al formato adentro de las llaves siguiendo la siguiente convención:

- Usamos la letra 'd' para indicar que la variable es un entero con signo.
- Usamos la letra 'f' para indicar que la variable es un float.
- Usamos la letra 's' para indicar que la variable es un string.

Por ejemplo:

```
In [10]: from math import e
```

```
In [11]: print('La constante de Euler es aproximadamente igual a {constante:1.10f}'
```

```
La constante de Euler es aproximadamente igual a 2.7182818285
```

Podemos obligarlo a imprimir ceros adelante usando `05d`, por poner un ejemplo.

```
In [12]: for k in range(10):
             print('Estamos en la posición {variable:05d}'.format(variable = k))
```

```
Estamos en la posición 00000
Estamos en la posición 00001
Estamos en la posición 00002
Estamos en la posición 00003
Estamos en la posición 00004
Estamos en la posición 00005
Estamos en la posición 00006
Estamos en la posición 00007
Estamos en la posición 00008
Estamos en la posición 00009
```

4 Leer y escribir archivos de texto plano

Antes de leer y escribir archivos, vale la pena saber en qué directorio (i.e. carpeta) estamos. Para hacerlo usamos los comandos `pwd` (que significa "print working directory") y `cd` (que traduce "change directory"), de ser necesario.

```
In [13]: pwd
```

```
Out[13]: '/home/mgd/Dropbox/AGL/cursillo_python/Notebooks_clases'
```

```
In [14]: ls # Para conocer los archivos que hay en el directorio
```

```
Clase 1.ipynb  Clase 3 - Final.ipynb  textos/
Clase 2.ipynb  Clase 3.ipynb          Untitled.ipynb
```

```
In [15]: cd textos/
```

```
/home/mgd/Dropbox/AGL/cursillo_python/Notebooks_clases/textos
```

```
In [16]: pwd
```

```
Out[16]: '/home/mgd/Dropbox/AGL/cursillo_python/Notebooks_clases/textos'
```

```
In [17]: ls
```

```
ejemplo_csv   ejemplo_de_escritura_por_lineas.txt  ejemplo_de_lectura.txt
ejemplo_csv~  ejemplo_de_escritura.txt             tabla_en_latex.txt
```

4.1 Leer archivos

Para escribir y leer archivos usamos las función `open`:

```
In [18]: open?
```

```
In [19]: ejemplo = open('ejemplo_de_lectura.txt')
```

```
In [20]: print(ejemplo.read())
```

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

```
In [21]: print(ejemplo.read()) # Una vez "ejemplo" se lee, queda vacío.
```

```
In [22]: ejemplo = open('ejemplo_de_lectura.txt')
         lineas = ejemplo.readlines()
         for linea in lineas:
             print(linea)
```

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod

tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,

quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo

consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse

cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non

proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

notamos que los strings que leemos usando `read` o `readlines` traen ciertos caracteres especiales (por ejemplo `\n`). Éstos se llaman *literales del string*. Entre otros están:

- `\\` backslash.
- `\'` comilla simple.
- `\"` comilla doble.
- `\n` nueva línea.
- `\t` tabulación.
- `\r` retorno de carro.

Por ejemplo:

```
In [23]: print('Hola \t Mundo')
```

```
Hola	 Mundo
```

Usualmente (y a la hora de procesar archivos) necesitamos quitar estos caracteres especiales. Para hacerlo usamos el método `replace` para strings.

4.2 Proyecto: encontrar la letra más común en un libro

Supongamos que tenemos un archivo `libro.txt` que contiene todo un libro y necesitamos saber cuál es la letra más común que aparece en él. Escribamos una función que haga precisamente eso.

Para hacerlo, dividimos la tarea en varias partes:

1. Necesitamos abrir el archivo y extraer el string con todo el contenido.
2. Necesitamos limpiar este string, quitando los literales del string.
3. Necesitamos contar la cantidad de letras y almacenar los resultados en alguna variable.
4. Necesitamos devolver el resultado al usuario.

```
In [32]: ls
```

```
DUBLINERS.txt                    THETRAGEDYOFROMEOANDJULIET.txt
THEBROTHERSGRIMMFAIRYTALES.txt   THUSSPAKEZARATHUSTRA.txt
THETIMEMACHINE.txt
```

```
In [33]: def contadordeletras(path_del_libro):
             # Primero abrimos el archivo y almacenamos el string
             libro_archivo = open(path_del_libro)
             libro = libro_archivo.read()

             # Limpiamos el string
             libro = libro.replace('\n', '')
             libro = libro.replace('\r', '')
             libro = libro.lower()

             # Contamos las letras
             diccionario_de_letras = dict()
             for letra in libro:
                 if letra.isalpha():
                     if letra in diccionario_de_letras:
                         diccionario_de_letras[letra] += 1
                     else:
                         diccionario_de_letras[letra] = 1

             # Devolvemos el resultado
             return diccionario_de_letras
```

Ya tenemos implementada la función. Intentémosla en *Así habló Zarathustra* de F. Nietzsche (en inglés)

```
In [34]: path = 'THUSSPAKEZARATHUSTRA.txt'
         print(contadordeletras(path))
```

```
{'y': 8792, 's': 24889, 'r': 21721, 'u': 12035, 'i': 25735, 'p': 5774, 'f': 7847, '
```

Sería muy interesante poder organizar las llaves de un diccionario por su valor. Después de una breve búsqueda en internet, notamos que podemos usar la librería `operator` y el siguiente pedazo de código:

```
In [35]: import operator

         diccionario = contadordeletras(path)

         lista_ordenada = sorted(diccionario.items(), key = operator.itemgetter(1))
         lista_ordenada.reverse()
         print(lista_ordenada)
```

```
[('e', 50291), ('t', 39313), ('a', 30973), ('o', 29789), ('h', 29357), ('n', 25885)
```

En *Así habló Zarathustra* aparece más la 't' que la 'a'.

4.3 Escribir archivos

```
In [57]: open?
```

Como podemos ver al preguntarle a IPython sobre la función `open`, podemos escribir sobre archivos dependiendo de la bandera que le pasemos:

- `'w'` para escribir sobre el archivo. Usar `'w'` resulta en sobreescribir.
- `'x'` para crear un archivo nuevo y abrirlo.
- `'a'` para escribir al final de un archivo existente.
- `'r+'` para tanto leer como escribir

```
In [36]: archivo_nuevo = open('ejemplo_de_escritura.txt', 'x')
```

Usando el método write, podemos escribir en el archivo usando el método `write`:

```
In [37]: archivo_nuevo.write('Hola mundo')

Out[37]: 10

In [38]: archivo_nuevo.close()
```

Análogamente, podemos escribir archivos usando `writelines`:

```
In [39]: segundo_archivo = open('ejemplo_de_escritura_por_lineas.txt', 'x')
         lista_de_filas = ['Hola, mundo\n', 'mi nombre es Miguel']
         segundo_archivo.writelines(lista_de_filas)
         segundo_archivo.close()

In [40]: sobre_escribir = open('ejemplo_de_escritura.txt', 'w')
         sobre_escribir.write('Aquí estoy sobreescribiendo lo que estuviera en el a
         sobre_escribir.close()

In [41]: no_sobre_escribir = open('ejemplo_de_escritura.txt', 'a')
         no_sobre_escribir.write('\nPero ahora no.')
         no_sobre_escribir.close()
```

4.4 Una froma más eficiente de leer el archivo

Para evitar tener que cerrar el archivo una vez terminamos con él, solemos usar la palabra clave `with`:

```
with open('archivo.txt') as archivo:
    # ...
```

5 Archivos CSV

Archivos CSV (comma separated values) son bastante comunes: pueden representar tablas, matrices... Podemos crear un archivo csv desde excel, por ejemplo.

Existe una librería de python para procesar archivos csv, y se llama precisamente `csv`:

Para usar los métodos de la librería `csv`, necesitamos un archivo de tipo io (como los que hemos estado importando).

```
In [50]: ls

ejemplo_csv   ejemplo_de_escritura_por_lineas.txt  ejemplo_de_lectura.txt
ejemplo_csv~  ejemplo_de_escritura.txt             tabla_en_latex.txt


In [51]: import csv

In [52]: with open('ejemplo_csv') as ejemplo:
             lista = csv.reader(ejemplo, delimiter=' ')
             print(list(lista))
             print(list(lista)) # Como siempre, se puede leer una sola vez nada más

[['1', '2', '3'], ['4', '5', '6'], ['7', '8', '9']]
[]
```

el objeto creado por `csv.reader` se colapsa a una lista de listas, en donde cada lista interna representa una fila del archivo original.

5.1 Proyecto: pasar de csv a tabla de LaTeX

Escribamos un script en python que nos permita obtener información sobre una tabla en csv para posteriormente escribir un archivo de texto plano con cómo luciría la tabla escrita en LaTeX. Recordamos que las tablas en LaTeX se hacen separando cada columna por un & y terminando la fila con un doble backslash. Para hacer esto, seguimos los siguientes pasos:

1. Comenzamos ingresando el archivo como un objeto io con `with open(path_del_archivo) as archivo:`.
2. Creamos un archivo nuevo para escribir sobre él con `open(path_del_archivo_nuevo, 'x')`.
3. Iteramos sobre cada fila del archivo original, operamos el contenido y escribimos sobre el archivo nuevo.

```
In [58]: import csv
         def tablaLaTeX(path_del_archivo, delimitador=';'): # excel por defecto.
             with open(path_del_archivo) as archivo:
                 try:
                     nuevo_archivo = open('tabla_en_latex.txt', 'x')
                 except:
                     nuevo_archivo = open('tabla_en_latex.txt', 'w')
                 for fila in csv.reader(archivo, delimiter = delimitador):
                     for k, elemento in enumerate(fila):
                         # Si el elemento no es final, lo escribimios elemento &
                         if k != len(fila) - 1:
                             nuevo_archivo.write(str(elemento) + '&')
                         else:
                             nuevo_archivo.write(str(elemento) + r'\\' + '\n')
                 nuevo_archivo.close()

In [59]: tablaLaTeX('ejemplo_csv', ' ')
```

6 Ejercicio

El ejercicio de la clase de hoy consiste en usar la función `contadorpalabras` en alguno de los textos que se analizaron en clase. Para hacerlo, debemos reimplementarla para sustituir ciertos caracteres especiales.

6.1 Primer ejercicio

Escriba una función `limpiador(texto)` que tome un string y elimine de él los siguientes literales del string y caracteres de unicode:

- `\n`
- `\r`
- `\u2018` (comilla simple izquierda)
- `\u2019` (comilla simple derecha)
- `\u0027` (apóstrofe)
- `\u201C` (comilla doble izquierda)
- `\u201D` (comilla doble derecha)
- Todos los signos de puntuación (al menos los que vienen de la librería `string`).

y, por último, retorne el string todo en minúsculas.

6.2 Segundo ejercicio

Escriba una función `contador_de_palabras(path_del_archivo)` que tome solo el *path* al archivo .txt y que realice lo siguiente:

1. Extraiga el contenido del archivo usando `open` y `read`.
2. Lo limpie usando la función auxiliar `limpiador` que escribimos en el primer ejercicio.
3. Cuente las palabras y almacene los resultados en un diccionario.

4. Genere una lista en donde se organicen los ítems del diccionario por sus valores (usando la librería `operator`, por ejemplo)
5. Devuelva ésta lista (o el reverso de ésta, para que estén de mayor a menor)

6.3 Tercer ejercicio

La ley de Zipf afirma que si en un texto la palabra más común ocurre n veces, la segunda palabra más común aparecerá $n/2$ veces, la tercera palabra más común aparecerá $n/3$ veces y así. ¿Qué tanto cumplen la ley de Zipf los 5 libros que descargamos?