Actividad 05 - *Uso de archivos, manejo de errores.*

Programación en Python - IAI - ECyT - UNSAM

1er cuatrimestre 2020

Cuando cerramos un programa y lo volvemos a usar un tiempo después esperamos que al abrirlo el programa esté en el mismo estado en el que lo dejamos, incluyendo los datos con que trabajábamos. A esta propiedad se la llama *persistencia* y la encaontramos en la agenda del teléfono, los sistemas bancarios y de exploración espacial. Ocasionalmente necesitamos transferir datos de un programa a otro para su análisis o para que otro usuario pueda acceder a ellos. Vamos a ver entonces, como usar archivos en Python para lograr persistencia. Encontrarán los programas y archivos necesarios en el GDrive de la materia.
Recuerden enviar los problemas resueltos a `python@unsam.edu.ar`

> Precaución: En estos ejercicios van a escribir archivos en el disco. Sepan que alterar archivos que son parte del sistema operativo u otros programas útiles, puede convertir a esos programas en inútiles. Tengan cuidado con los nombres de los archivos que van a modificar.
>
> Si un programa deja de correr y deja archivos abiertos, el sistema operativo se negará a borralos. Para poder borrarlos necesitan cerrar la consola de Python asociada al programa dueño del archivo.

Archivos

Por favor leer la Unidad 11 *Manejo de archivos* de "Algoritmos y Programación" (p.147) y compruebe que efectivamente los ejemplos se cumplen en su intérprete. Para las pruebas, elija cualquier archivo de texto (también llamado `ascii` ó `.txt`). Encontrará archivos `'.txt'` en el GDrive de la materia.

Comprenda como funciona el `Codigo 11.1` (*log.py*) (p. 150) y luego haga el siguiente ejercicio:

1. Escriba un programa que, usando el Código 11.1 del libro:
   - Le pida un nombre de archivo.
   - Abra el archivo.
   - En un ciclo, le pregunte frases o palabras y las guarde en el archivo
   - Si la frase es `'*'` (asterisco) termine, cerrando el archivo primero.
   - Luego usted, a mano y usando un editor de texto, abra dicho archivo y compruebe que su contenido es efectivamente lo que usted esperaba.
2. Ejecute el programa del ejercicio anterior y verifique, mirando el directorio en el que se está guardando su archivo, que el archivo tiene tamaño cero `(0 bytes)` hasta que lo cierra con `'*'`; y que el sistema operativo no le permite modificarlo (p. ej. moverlo) hasta que lo haya cerrado en Python.

   Nota 1: En el entorno Spyder, en el File Explorer (junto al Variable Explorer) encontrará donde está trabajando su programa. Sino, puede dar el nombre completo.
3. Escribir una función, llamada `Encabezado()` que reciba el nombre de un archivo de texto y un número N y devuelva las primeras N líneas del archivo. *Observe* que, una vez leídas esas N primeras líneas, para volver a leerlas necesita hacer un `seek()` o cerrar y volver a abrir el archivo.

4. Escribir una función, llamada `Copiar()` que reciba nombres de dos archivos (uno existente, sea de texto o binario, y el otro a ser creado) y copie todo el contenido del primero al segundo. *Notar que todo archivo de texto puede tratarse, además, como archivo binario.*

5. (*) Escribir una función, llamada `ContarPartes()`, que dado el nombre de un archivo de texto, lo procese e imprima cuantas líneas, cuantas palabras y cuantos caracteres contiene el archivo.

6. Escribir una función, llamada `Buscar()`, que reciba una cadena y el nombre de un archivo e imprima únicamente las líneas del archivo que contienen la cadena recibida.

7. (*) Escribir una función, llamada `CifrarArchivo()` que reciba el nombre de un archivo de texto de origen y uno de destino, de modo que para cada línea del archivo origen, se guarde una línea cifrada en el archivo destino. El algoritmo de cifrado a utilizar es (si !!) Cifrado César de la Guía 01.

8. (*) Escribir por supuesto la función complementaria que sea capaz de des-cifrar un archivo cifrado. `DesCifrarArchivo()`

9. Persistencia de un diccionario:
   Imagine un archivo de texto que guarda un diccionario, con el siguiente formato:

```
clave1, valor1.1, valor1.2, .... valor1.N
clave2, valor2.1, valor2.2, .... valor2.N
.
.
claveN, valorN.1, valorN.2, .... valorN.N
```

   a) Escribir una función llamada `Dicc_Arch()` que reciba un diccionario y un nombre de archivo, y guarde el contenido del diccionario en el archivo, con el formato especificado arriba.

   b) Escribir una función complementaria `Arch_Dicc()` que reciba un nombre de archivo, y devuelva un diccionario con el primer campo de cada línea como clave y los campos siguientes como valores de esa clave.

10. Rescate su vieja agenda (modificada en el ejercicio 4 de la Guía 04) y haga la mínima cantidad de modificaciones necesarias para que el programa lea el diccionario de un archivo al inicio y que, cuando usted le pida terminar (`*`) guarde el diccionario en ese archivo. *Nota:encontrará una agenda MUY modificada en el GDrive de la materia, puede ayudarse con ésa, pero aquí le pedimos las mínimas modificaciones necesarias.*

Manejo de errores y excepciones

Leer la Unidad 12 del libro (p. 162). Es importante comprender íntegramente el concepto de manejo de excepciones. Luego:

11. Proteja su agenda para que el programa se comporte elegantemente aún si el archivo con el diccionario no existe al intentar leerlo. *Elegantemente significa: no le dé un error críptico al usuario, déle información que lo ayude a solucionar el problema*

12. Copie a mano el archivo diccionario a un pendrive o cualquier otro dispositivo removible. Ahora: Modifique su `Agenda` para que, si el pendrive no existe al momento de grabar el archivo y salir, el programa ofrezca una solución alternativa y no pierda los datos (p. ej. pida un nuevo nombre completo de archivo donde guardar los datos).

Procesamiento de archivos

Lean por favor la Unidad 13 del libro (p. 174) Este capítulo es una introducción al procesamiento de archivos para combinar y obtener grandes cantidades de datos ordenados de modo tal que el análisis resulte simple. El objetivo es organizar racionalmente los datos para simplificar el análisis.

Ejercicios con libros reales y sus palabras, como fuente de gran cantidad de datos.

Algunas herramientas útiles:

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

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

>>>"... un ejemplo, las comas al final se van,.".strip(' ,.')
"un ejemplo, las comas al final se van"
```

13. Escriba una función que lea el archivo `palabras.txt`, limpie las palabras de signos de fin de línea (recuerde, aunque no los vea, `\r\n`, p. 148) y las guarde en una lista llamada `palabras`.

14. Calcule y compare el porcentaje de palabras en el archivo `palabras.txt` (usando (13)) que contienen cada una de las vocales. (ojo con los acentos!).

15. 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 de la materia 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.

16. 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.

17. (*) Imprima las 20 palabras más utilizadas en el texto.

18. (*) Escriba un programa que lea su archivo favorito palabra por palabra e imprima aquellas palabras que no están en la lista llamada `palabras` creada en (13). ¿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).