



Práctica 7: Tipos de datos estructurados en Python

1. Objetivos

- Aprender a hacer programas que utilicen los diferentes objetos de datos estructurados que ofrece Python: cadenas, tuplas, listas y diccionarios.
- Conocer y utilizar los métodos (funciones) asociados a los datos estructurados en Python.
- Consolidar el desarrollo de programas en Python siguiendo las directrices de la programación modular utilizando funciones.
- Practicar la interacción de los programas con archivos de texto para leer y escribir datos.

2. Introducción

Esta práctica consiste en el desarrollo de un programa para hacer un análisis estadístico simple de las palabras utilizadas en diferentes obras literarias. El programa debe analizar varios libros para encontrar las palabras más utilizados y los autores con un léxico más amplio.

2.1. Interfaz de usuario

La interacción del programa con el usuario es bastante simple. Al inicio, el programa debe pedir los nombres de los archivos de texto a procesar o buscarlos automáticamente con la opción `auto`.

Menú principal

```
Análisis estadístico de vocabulario en obras literarias
```

```
Ingrese los nombres de los archivos a procesar (separados por espacio) o auto  
para el modo automático:
```

```
book1.txt book2.txt book3.txt book4.txt
```

Resultado del análisis estadístico

```
Los archivos analizados fueron: book1.txt, book2.txt, book3.txt y book4.txt
```

```
En los 4 libros analizados se utilizaron 3987 palabras diferentes así:
```

```
Romeo y Julieta: 3423
```

```
Cita con la Muerte: 2456
```

```
La conquista de la felicidad: 2798
```

```
Emma: 3309
```

El autor con vocabulario más amplio es William Shakespeare en Romeo y Julieta.

Las 30 palabras más usadas en el acumulado de palabras son:

the, and, or, start, talk, with, upon, house, horse, tree, park, walk, on, off, see, knife, know, listen, without, who, fast, father, love, run, he, they, she, we, are, is

Y luego de filtrar artículos, pronombres, preposiciones y conjunciones, la lista es:

start, talk, house, horse, tree, park, walk, see, knife, know, listen, fast, father, love, run, are, is, ugly, small, big, cake, sad, train, river, clean, city, pass, bridge, eat, hate

2.2. Requerimientos del programa

Requisitos funcionales

El programa desarrollado debe abrir los archivos de texto indicados y procesarlos de manera que muestre en pantalla al usuario la información que se indicó en la sección anterior.

La opción `auto` que se menciona en el menú principal debe llevar a que el programa determine los archivos con extensión `.txt` que hay en la carpeta `books` y utilizar dichos libros para hacer el análisis.

Si lo prefiere, utilice la siguiente lista de artículos, pronombres, preposiciones y conjunciones para ser excluidos en el último listado: *and, the, a, i, to, it, of, he, in, you, that, but, so, on, for, up, we, me, him, they, with, out, then, his, as, them, she, down, when, by, at, my, or, what, her, this, off, over* .

Requisitos no-funcionales

El programa desarrollado debe cumplir con las siguientes condiciones además de funcionar correcta y eficientemente:

- Crear al menos 5 funciones a criterio del programador con sus respectivas especificaciones a manera de docstrings.
- Crear dos módulos (archivos fuente), uno para el programa principal y otro para las funciones.
- Utilizar strings, tuplas, listas y diccionarios
- El menú principal debe hacer una única pregunta al usuario tal como se muestra arriba.

2.3. Desarrollo paso a paso

En este, como en cualquier otro programa, es importante seguir una metodología de trabajo ordenada y por partes, de manera que la complejidad del problema no impida llegar a su solución. Luego de asegurarse que entiende lo que hay que hacer, al empezar a pensar cómo hacerlo es muy importante ir por partes que poco a poco se irán uniendo para obtener el resultado final.

A continuación se sugiere una lista no necesariamente completa de subproblemas que puede ir abordando, uno por uno, que le servirán para la implementación del programa completo.

- Abra un archivo que contenga un texto corto de prueba y cree una lista con todas las palabras del archivo convertidas a minúsculas y excluyendo espacios y signos de puntuación.
- Abra uno de los archivos de la carpeta books y aplique el procedimiento anterior solo a partir del inicio real del libro que se puede identificar por el texto `**** START OF THIS PROJECT GUTENBERG EBOOK...` y hasta el final del libro identificado por `**** END OF THIS PROJECT GUTENBERG EBOOK...`.
- Al proceso anterior, agregue la capacidad de encontrar el título y el autor del libro y guárdelo en variables.
- Construya un diccionario a partir de la lista de palabras para ir anotando las veces que se encuentra cada palabra.
- Construya una lista de tuplas a partir del diccionario para luego ordenarla y así determinar las palabras más usadas.
- Construya manualmente una lista de las palabras que quiere excluir de la lista y genere una copia de lista anterior aplicando el filtro.
- Repita el proceso para varios libros, entregando los resultados como lo pide el programa.
- Reciba lo que el usuario ingrese en el menú principal y detecte en primera instancia si deberá ir a la carpeta books a buscar todos los libros que haya, o si en cambio el usuario ha ingresado una lista de archivos específica.
- Consulte cómo listar desde Python todos los archivos que hay en una carpeta y construya una lista solo con los nombres de aquellos archivos con extensión .txt.

2.4. Pruebas

Dada la importancia del proceso de pruebas funcionales de un programa y el tiempo que estas pueden consumir, con frecuencia es necesario automatizarlas, es decir, crear un programa que le haga pruebas a otro programa. A nivel de función, esto consiste simplemente en invocar la función varias veces utilizando datos de entrada escogidos estratégicamente de manera que se puedan revelar posibles errores.

Luego de terminar el desarrollo del programa propuesto en esta práctica, desarrolle un programa adicional, en un archivo fuente aparte, que invoque las funciones del programa para hacerles pruebas. Haga pruebas de caja blanca y de caja negra tratando de revelar errores y explique al profesor el porqué de cada conjunto de entradas de prueba que utilizó.

2.5. Opcional

Haga que el programa genere al final un párrafo de 100 palabras, donde cada palabra sea generada al azar pero con una probabilidad que estará determinada por la cantidad relativa de veces que aparece dicha palabra en todos los textos analizados. Por ejemplo, si solo hubiera dos palabras ("the" y "river"), donde "the" aparece 30 veces y "river" aparece 20, cada que se genere una palabra al azar, deberá hacerse con una probabilidad del 60% para "the" y 40% para "river". De esa manera, el párrafo resultante tendrá aproximadamente la misma distribución estadística de los textos analizados.

3. Evaluación

La evaluación se basará en los códigos enviados y una sustentación oral sobre los temas de la práctica. Además, se tendrá en cuenta una bonificación para quien haga la parte opcional.