PYTHON PARA LINGÜISTAS

LECTURA Y ESCRITURA DE FICHEROS

CODIFICACIONES

ALEJANDRO ARIZA

CENTRE DE LLENGUATGE I COMPUTACIÓ

UNIVERSITAT DE BARCELONA

¿QUÉ SABEMOS HASTA AHORA?

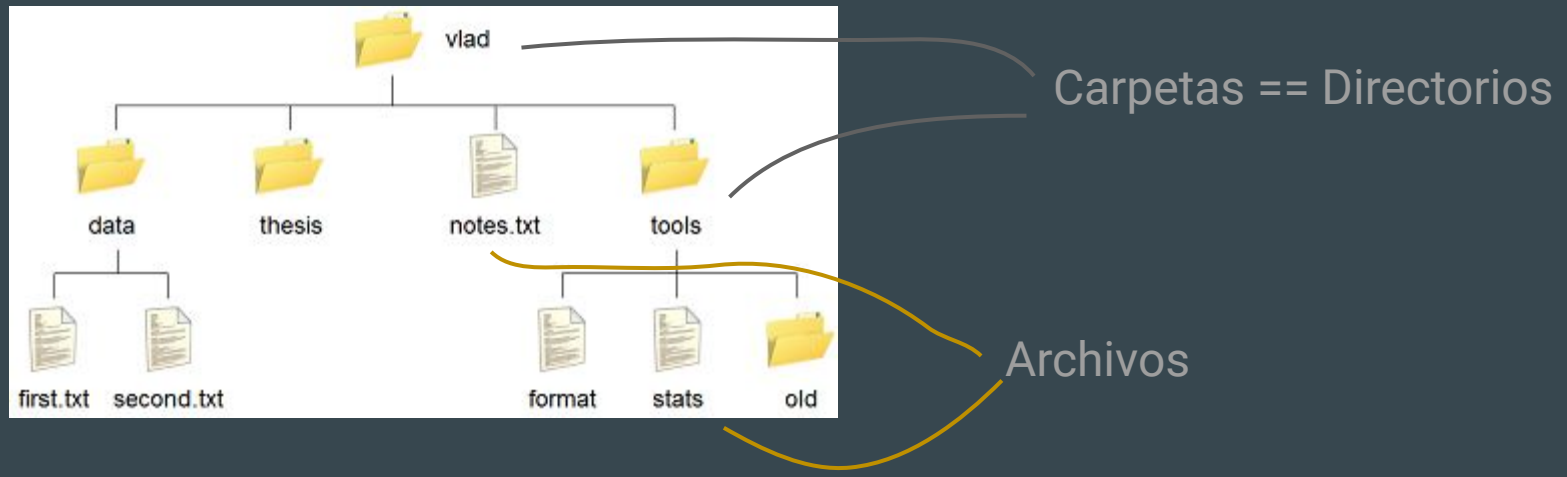
- Tipos de datos (string, int, float, Boolean, list, dictionary)
- Estructuras de decisión (if, elif, else)
- Bucles (while, for)
- Pedir datos al usuario (input)
- Funciones y métodos

TRABAJANDO CON FICHEROS

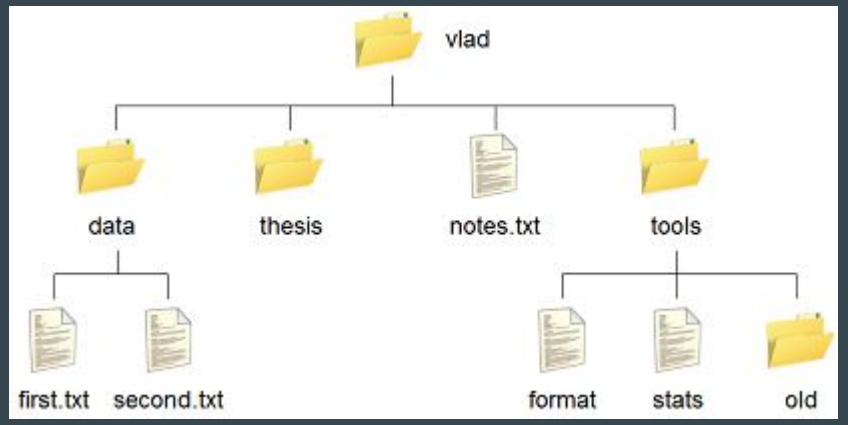
- De forma bastante frecuente, un programa necesita trabajar con ficheros.
- Por lo tanto, necesita ser capaz de leer estos ficheros (e.g.: un corpus)
- También, el programa necesita ser capaz de escribir información en un fichero (e.g.: guardar variables, logs, un corpus modificado)

EL SISTEMA DE ARCHIVOS: ARCHIVOS Y CARPETAS

- El sistema de archivos es una capa lógica que organiza la información que se guarda en el disco en una jerarquía de carpetas con un nombre que le permita recuperarlas.



EL SISTEMA DE ARCHIVOS: ARCHIVOS Y CARPETAS (2)



Los ficheros tienen una extensión: `.txt`, `.xml`
Son solo una convención, forman parte del nombre e indican el "formato" en el que se guardan los datos.

Rutas: rama de la jerarquía que va desde la raíz (vlad/) hasta el archivo o carpeta de interés.
Ruta relativa: es una ruta cuya raíz no es la raíz del sistema (C:/) sino la carpeta en la que nos encontremos. Se representa: "./"

RUTA EN EL SISTEMA DE ARCHIVOS Y JERARQUÍA

- Dependiendo del sistema operativo, la jerarquía cambia:
  - Windows: C:/, Program Files...
  - Mac y Linux: /home, /bin...
- Dos formas de definir una ruta:
  - Absoluta: Desde la raíz del sistema de archivos. 'D:/python_course/notebooks/macbeth.txt'
  - Relativa: desde la carpeta donde nos encontramos. './macbeth.txt'
- Para este curso, guardaremos los ficheros que necesitemos en la misma carpeta que nuestro código para que podamos usar rutas relativas tan simples como 'macbeth.txt' (el prefijo "./" es opcional)

CONTENIDO DE UN FICHERO

- ¿Qué hay dentro de los ficheros?
- ¿Como podemos pasar de un fichero de texto a listas y string?

- Para este curso, nos centraremos en ficheros de texto → ficheros cuyo contenido está preparado para leerse como secuencias de caracteres → txt, csv, xml
- Por ahora, asumiremos que cada fichero únicamente guarda strings de texto.

TRABAJANDO CON FICHEROS

- En programación, trabajar con ficheros implica las siguientes operaciones:
  - **Open**: el programa accede al fichero y se prepara para usarlo.
  - **Read**: el programa lee el contenido del fichero, por lo general de forma secuencial.
  - **Write**: el programa escribe datos al final del fichero.
  - **Close**: El programa se asegura de que todas las operaciones fueron ejecutadas en el sistema de archivos y libera el fichero.

TRABAJANDO CON FICHEROS EN PYTHON

- Abrir fichero en Python tiene la siguiente sintaxis:

  ```
  with open("filename.txt","r") as f:
  ```

- Antes de poder trabajar con el contenido del fichero es necesario abrirlo.
- "filename.txt" es el nombre del fichero que queremos abrir (en realidad es la ruta relativa al fichero que queremos abrir).
- El segundo parámetro de la función open() es el modo en que lo queremos abrir. En el ejemplo, nos encontramos la etiqueta "r" que significa lectura (READ); para la escritura, usaríamos "w"
- f – es una variable de tipo "file" con la que podemos utilizar métodos específicos.

MÉTODOS DEL TIPO DE DATO “FILE”

- text_1 = f.read() – Lee el fichero entero y lo guarda dentro de una variable de tipo string llamada “text_1”.
- line_1 = f.readline() – Lee la siguiente “línea” y la guarda en una variable de tipo string. El programa recuerda cuál fue la última línea leída y devuelve la siguiente.
- lines = f.readlines() – Lee el fichero entero y lo guarda en una lista de strings donde cada elemento es una línea del fichero.
- f.write(string_1) – Escribe un string en un fichero.
- Los ficheros están organizados en líneas. Un símbolo especial (depende del sistema operativo) determina la finalización de una línea y comienzo de la siguiente.

LECTURA DE FICHEROS EN PYTHON

- ¿Cómo podemos leer un fichero en Python?
- f.read() – extrae el contenido del fichero en un string.

  El string puede ser muy grande dado que no hay segmentación de líneas, párrafos o capítulos. Estos últimos pueden ser difíciles de recuperar.

- f.readlines() – extrae el contenido del fichero en una lista de strings.

  La lista guarda frases, párrafos, capítulos (lo que contenga cada línea del fichero). También puede ser muy grande.

- ¿Cuál es la forma óptima de leer el contenido de un fichero?

LECTURA DE FICHEROS EN PYTHON

- La forma típica de lectura de ficheros es línea por línea usando un bucle for:

```
with open(path, 'r') as f:
    for line_1 in f:
```

- Este código lee línea por línea de forma que en cada iteración podemos procesarlas por separado:
  - Separar los tokens
  - Modificar un diccionario
  - Contar frecuencias

ESCRITURA DE FICHEROS EN PYTHON

- Hay múltiples formas de escribir ficheros en Python. A continuación, tenéis dos formas de hacerlo:

```
with open(path2, 'w') as f2:
    f2.write(string_1)
```

<- Escribe un string al final del fichero utilizando el método de la variable f2

```
with open(path2, 'w') as f2:
    print(string_1,file=f2)
```

<- Usando la función print(), podemos especificar donde imprimir el string

CODIFICACIÓN DE DATOS

- El ordenador únicamente entiende 0s y 1s.
- Es suficientemente sencillo para números → Codificación en base binaria en vez de base 10:
  - 0 → 000; 1 → 001; 2 → 010; 3 → 011; 4 → 100; 5 → 101 ...
- Sin embargo, ¿Qué hacemos con el texto? Necesitamos algún código que transforme el texto a números para poder codificarlos de forma binaria:
  - a → 1; b → 2; c → 3; d → 4; e → 5; f → 6 ...
- Esta transformación recibe el nombre de codificación y funciona de la siguiente forma con texto:
  - “don’t stop”:

d	o	n	‘	t		s	t	o	p
4	15	14	27	20	0	19	20	15	16

CODIFICACIÓN DE TEXTOS

- Existe un problema: diferentes alfabetos, símbolos especiales, etc.
- Como resultado, tenemos muchas codificaciones diferentes.
- Cada codificación transforma diferentes conjuntos de caracteres.
- Algunas codificaciones usan diferentes números para los mismos caracteres (e.g. “a” – 1 en una, “a” – 101 en otra)
- Algunas codificaciones usan los mismos números para diferentes caracteres (e.g.: “a” – 1 en una, mientras que “α” es 1 en otra)
- Diferentes sistemas operativos (windows, mac, linux) usan diferentes codificaciones por defecto.

UNICODE

- Para resolver estos conflictos entre diferentes codificaciones, existe la codificación Unicode.
- La codificación Unicode intenta tener una codificación “universal” para todos los caracteres.
- Puede ser utilizada para todos los lenguajes (e.g.: Latin, Slavic, Asian).
- UTF-8 es la codificación Unicode más popular.
- Para utilizar una codificación Unicode, necesitamos pasarlo como argumento de entrada a la función open():

```
with open('macbeth.txt', 'r', encoding='utf-8') as f:
```

STRINGS Y CARACTERES ESPECIALES

- Los strings pueden incluir algunos caracteres que se comporten de una forma específica.
- Por ejemplo, las comillas indican el comienzo y final del string. ¿Cómo podemos usar las comillas dentro del string?
- El operador especial “\” (“escape” character) nos permite tratar caracteres especiales como texto.
- El siguiente string es válido: ‘una comilla \’ dentro de un string’
- Si quereís imprimir el operador “\” podéis escribirlo dos veces: ‘así \\’ (solo lo imprimirá una vez)

STRINGS Y CARACTERES ESPECIALES (2)

- El carácter “\” también puede crear algunos operadores especiales cuando se combina con ciertas letras.
- “\n” – Esto es un símbolo especial que delimita el final de una línea en un string o fichero.
- “\t” – Esto indica un espacio de tabulación en un string o fichero.
- En la práctica, probad a imprimirlos y ved el resultado.

¡GRACIAS!
¿PREGUNTAS?