Repaso de conceptos básicos
Ficheros

Fundamentos de Programación

Febrero 2019

Ficheros: ideas generales

- Leer y escribir en un fichero es más lento que hacerlo en memoria (aunque cada vez menos: SSD).
- ¿Por qué usarlos entonces? Por la **persistencia** de los datos, es decir, porque lo que escribimos en un fichero permanece almacenado en la máquina incluso después de apagarla.
- Un fichero es un almacén de datos, que podemos interpretar de muchas formas: por ejemplo, en bloques de 1 byte codificados como enteros con signo (rango=[-128:127]), en bloques de 8 bytes codificados como números de coma flotante, etc.
- En este curso manejaremos principalmente ficheros de texto, que guardan una secuencia de caracteres, utilizando una codificación concreta (ASCII, latin-1, UTF-8, etc.).

Ficheros: ideas generales

- Mediante un fichero de texto podremos representar cualquier tipo de dato.
- Será imprescindible establecer un mecanismo para recuperar exactamente el dato original a partir de su representación, de modo que podamos reconstruir en memoria una base de datos almacenada en un fichero.
- En los últimos años, se tiende a representar datos mediante ficheros de texto en **formato XML**, que incluye explícitamente toda la meta-información necesaria.
- **Ventajas:** el fichero es legible y portable (sin más que conocer su codificación, que en el caso de ficheros XML se suministra).
- **Desventajas:** representar datos mediante cadenas de caracteres es muy ineficiente en cuanto al tamaño de la representación y ralentiza las operaciones de lectura y escritura.

Ficheros: ideas generales

Ejemplo de fichero XML

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
    <display-name>HelloWorld Application</display-name>
    <description>
        This is a simple web application with a source code organization
        based on the recommendations of the Application Developer's Guide.
    </description>

    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>examples.Hello</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>

</web-app>
```

Ficheros en Python

- Para operar con un fichero se utiliza una variable auxiliar, que en Python se crea e inicializa mediante la función **`open()`**:

```
fp = open('/users/mikel/datos.txt','r',encoding='utf-8')
```

- Existen tres modos principales de apertura: lectura (`'r'`), escritura (`'w'`) y adición (`'a'`).
- Si se intenta abrir en modo `'r'` un fichero que no existe, se producirá un error.
- Si se abre en modo `'w'` un fichero existente, se sobre-escribirá (en modo `'a'`, se añadirán contenidos al final)
- Se debe dar el nombre completo del fichero, bien mediante la ruta completa desde el directorio raíz, bien mediante la ruta relativa desde el directorio donde se ejecuta el programa
- Es posible forzar una codificación mediante el parámetro `encoding` en la función `open()`. La codificación por defecto depende de la configuración del sistema.

Ficheros en Python

- Métodos más habituales:
  - `fp.read([n])` → cadena representando todo el fichero
  - `fp.readline([n])` → cadena con '\n' al final
  - `fp.readlines([size])` → lista de cadenas con '\n' al final
  - `fp.write(s)`
  - `fp.writelines(lines)`
  - `fp.seek(offset[,origen])` :: posicionamiento
  - `fp.flush()` :: volcado del buffer a disco
  - `fp.close()`

Ficheros en Python

Método str.format( ): convertir datos a cadena con formato

- Sintaxis: **s.format (argumentos)**
- Genera una cadena de caracteres, donde los argumentos que van entre paréntesis aparecen formateados de acuerdo a la especificación suministrada en s.
- Ejemplos:

```
>>> "Los {} costaron más que los {}".format("libros","discos")
'Los libros costaron más que los discos'
>>> "Los {1} costaron más que los {0}".format("libros","discos")
'Los discos costaron más que los libros'
>>> "Hola {name}: tienes {age} años".format(age=18,name="Luis")
'Hola Luis: tienes 18 años'
>>> x=3-4j
>>> "Parte real: {0.real}, Parte imaginaria: {0.imag}".format(x)
'Parte real: 3.0, Parte imaginaria: -4.0’
```

Ficheros en Python

Método str.format( ): convertir datos a cadena con formato

- Se hace referencia a los argumentos indicando entre llaves su posición: **{n}**
- Los argumentos inicializados en la propia llamada se pueden referenciar mediante el nombre: **{name}**
- Si el argumento es una lista, se puede acceder a sus elementos mediante indexado: **{n[i]}**; si se trata de un diccionario, mediante una clave: **{n[clave]}**; y si se trata de un argumento con atributos, mediante el nombre del atributo: **{n.atributo}**
- Ejemplos:

```
>>> d={'fecha':'febrero de 2019', 'nitems':36, 'precio':23.78}
>>> "{0[fecha]}: {0[nitems]} items, a {0[precio]} euros".format(d)
'febrero de 2019: 36 items, a 23.78 euros'
>>> lst=["verano",15,12.11,0.5,34,"helado"]
>>> "los dos primeros son {0[0]} y {0[1]}".format(lst)
'los dos primeros son verano y 15'
```

Ficheros en Python

Método str.format( ): convertir datos a cadena con formato

- En el caso de objetos indexados por enteros (como listas), sus elementos pueden convertirse en argumentos sin más que pasar el nombre del objeto con un asterisco por delante (***name**):

```
>>> lst=["verano",15,12.11,0.5,34,"helado"]
>>> "los dos primeros son {} y {}".format(*lst)
'los dos primeros son verano y 15'
```

- En el caso de objetos indexados por cadenas (diccionarios), sus elementos pueden referenciarse mediante la clave de acceso (como si fueran argumentos inicializados en la llamada) pasando el nombre del objeto con dos asteriscos por delante (****name**):

```
>>> d={'fecha':'febrero de 2019’, 'nitems':36, 'precio':23.78}
>>> "{fecha}: {nitems} items, a {precio} euros".format(**d)
'febrero de 2018: 36 items, a 23.78 euros'
```

Ficheros en Python

Método str.format( ): convertir datos a cadena con formato

- Tras la referencia posicional o nominal de un argumento y tras el carácter ':', se puede añadir una especificación de formato.
- Ejemplo:

```
>>> x=0.1428614
>>> y=0.5455237
>>> "X:{0:10.4f}, Y:{1:10.4f}, Etiq: {tag}".format(x,y,tag="Fuente")
'X:    0.1429, Y:    0.5455, Etiq: Fuente'
```

- Especificación de formato:

[[fill]align][sign][#][0][width][.precision][type]

- **fill**: carácter de relleno (por defecto, espacio)
- **align**: < (a izquierda), > (a derecha, por defecto), ^ (centro), = (a derecha, salvo el signo, que va a la izquierda)
- **sign**: '+', '-' (por defecto), ' ' (espacio)

Ficheros en Python

Método str.format( ): convertir datos a cadena con formato

- Especificación de formato (continuación):

  [[fill]align][sign][#][0][width][.precision][type]

- **#**: (sólo para enteros) añade el prefijo 0b para salida en binario, 0o para salida en octal y 0x para salida en hexadecimal
- **0**: equivale a usar '=' como alineamiento y '0' como relleno
- **width**: mínima anchura del campo
- **precision**: enteros: no está permitido; reales: indica el número de decimales; otros: indica la anchura máxima del campo
- **type**:

d,b,o,x,X,n	entero (n: igual que d, pero local)
f,F,e,E,g,G,n	real (n: igual que g, pero local)
%	porcentaje (x 100.0 y añade '%')
s	cadena de caracteres
c	carácter

Ficheros en Python

repr( ) y eval( ): escritura / lectura en ficheros de texto

- La función repr() genera una cadena que trata de representar el contenido del argumento, en el sentido de que pueda ser evaluado por la función eval(), que recuperaría el valor original.
- Veamos ejemplos en los que str() y repr() dan distinto resultado:

```
>>> s='Hola mundo\n'
>>> str(s)
'Hola mundo\n'
>>> repr(s)
"'Hola mundo\\n'"
>>> print(str(s))
Hola mundo

>>> print(repr(s))
'Hola mundo\n'
```

```
>>> import datetime
>>> today = datetime.datetime.now()
>>> str(today)
'2018-01-24 15:48:43.020935'
>>> repr(today)
'datetime.datetime(2018, 1, 24, 15, 48, 43, 20935)'
```

Ficheros en Python

repr( ) y eval( ): escritura / lectura en ficheros de texto

- Un dato representado mediante la función repr() puede ser recuperado mediante la función eval():

```
>>> a = ("Mikel",18,7.87)
>>> a_repr = repr(a)
>>> b = eval(a_repr)
>>> print(b)
('Mikel', 18, 7.87)
```

- Esto nos permite utilizar ficheros de texto para guardar bases de datos formadas por secuencias de ítems, a razón de un ítem por línea, suponiendo que todos los ítems tienen la misma estructura.
- **Procedimiento de escritura**:

  (1) si fuera necesario, agrupar todos los datos de un ítem en una tupla
  (2) usar repr() para obtener una representacion textual del ítem
  (3) escribir dicha representación en una línea del fichero
  (4) volver a (1) hasta terminar de procesar la base de datos

Ficheros en Python

repr( ) y eval( ): escritura / lectura en ficheros de texto

Procedimiento de lectura:

(1) leer una línea del fichero

(2) utilizar la función eval() para recuperar el ítem

(3) si se tratara de una tupla, asignar la salida de eval() al conjunto de variables adecuado

(4) volver a (1) hasta que no queden más líneas en el fichero

Ejemplo (E/L de una lista):

```
import random

def rndstr(n):
    s=""
    nl=ord('z')-ord('a')+1
    for i in range(n):
        s=s+chr(int(random.random()*1000)%nl+ord('a'))
    return s

# creación de la lista
l=[]
for i in range(5):
    tupla=(i,rndstr(10))
    l.append(tupla)

# almacenamiento de la lista en un fichero
fp=open("lista.txt","w")
for item in l:
    fp.write(repr(item)+'\n')
fp.close()

# reconstrucción de la lista desde el fichero
l_new=[]
fp=open("lista.txt","r")
for line in fp:
    tupla=eval(line)
    l_new.append(tupla)
fp.close()
```

Ficheros en Python

El módulo pickle: poner datos en conserva en un fichero

- El módulo `pickle` permite volcar un objeto cualquiera en un fichero, utilizando una representación (definida mediante un protocolo) que permite recuperarlo íntegramente.
- Métodos:

```
pickle.dump(objeto,fichero[,protocolo])
pickle.load(fichero)
```

- Por defecto, el protocolo de representación es el 3.
- El protocolo más simple es el 0 (compatible con todas las versiones anteriores de Python).
- La variable `pickle.HIGHEST_PROTOCOL` contiene el identificador del protocolo más reciente (de momento, el 4).
- Al recargar los datos, el protocolo se detecta automáticamente.

Ficheros en Python

El módulo pickle: poner datos en conserva en un fichero

- Ejemplo 1:

```
# x e y son variables reales
fp = open('datos','wb')
pickle.dump(x,fp)
pickle.dump(y,fp)
fp.close()

fp = open('datos','rb')
x = pickle.load(fp)
y = pickle.load(fp)
fp.close()
```

- Ejemplo 2:

```
# l es una lista de objetos
fp = open('datos','wb')
pickle.dump(l,fp)
fp.close()

fp = open('datos','rb')
l = pickle.load(fp)
fp.close()
```

Ficheros: ejercicios propuestos

1. Escribir un programa que pida al usuario una cadena con **caracteres prohibidos** y el nombre de dos ficheros de texto. El programa deberá escribir en el segundo fichero, para cada línea del primer fichero, el número de línea seguido de aquellas palabras de dicha línea que no contengan ninguno de los caracteres prohibidos.
2. Escribir un programa que pida al usuario una cadena con **caracteres permitidos** y el nombre de dos ficheros de texto. El programa deberá escribir en el segundo fichero, para cada línea del primer fichero, el número de línea seguido del porcentaje de palabras de esa línea que contengan sólo caracteres permitidos.
3. Escribir un programa que pida al usuario una cadena con **caracteres obligatorios** y el nombre de dos ficheros de texto. El programa deberá escribir en el segundo fichero las palabras del primer fichero que contienen todos los caracteres obligatorios, a razón de una palabra por línea y, junto a cada palabra, el número de veces que aparece.
4. Escribir un **primer programa** que genere una lista de 1000 cadenas de caracteres aleatorias (formadas sólo por letras minúsculas, de longitud entre 5 y 10), y la guarde en un fichero mediante el módulo **`pickle`**. Escribir un **segundo programa** que cargue dos de esos ficheros en sendas listas (también mediante el módulo **`pickle`**) y calcule la media del número de caracteres en común entre las palabras que ocupan las mismas posiciones en las dos listas. **¿Se podría estimar a priori esa media?**