Prácticas de programación en python
Servicios Telemáticos 2015-2016
Grado en Ingeniería Telemática
Universidad Rey Juan Carlos
`http://gsyc.urjc.es`

Observaciones

- Crea el directorio `~/st/practica01` donde escribirás el resto de ficheros de esta práctica. Recuerda que la recogida de las prácticas se hará automáticamente el dia del examen, así que es muy importante que respetes al pie de la letra todos los nombres de ficheros y directorios que indiquen todos los enunciados.

Práctica 1.1. Hola Mundo, listas

Vamos a empezar por un script muy sencillo que será poco más que un *hola mundo* sobre el uso de listas. Ponle como nombre `~/st/practica01/cola.py`.

1. Simula la lista de espera de clientes en la caja de un supermercado Define dos listas: `caja01` y `caja02`
2. Mediante sentencias básicas de python, vete añadiendo clientes (elementos) a `caja01` y `caja02`. Imprime de vez en cuando la lista
3. Un cliente desiste y abandona la cola
4. Un cliente tiene una compra muy pequeña y los demás le permiten ponerse el primero
5. Se abre la caja03. Los clientes de la caja02 pasan en orden a la caja03. Esto es, cambian de caja los que ocupan las posiciones segunda, cuarta, sexta... No uses bucles, simplemente escribe unas pocas líneas moviendo unos pocos elementos. No modifiques la caja01.
6. En un arrebato de injusticia, invierte el orden los clientes en la caja01.
7. Con una sentencia `if`, haz que si existe un cliente llamado Ernesto en caja01, pase a ser el primero de la cola. Muestra un mensaje que diga *Ernesto pasa de la posición n a la 1*, donde n es la posición que ocupaba inicialmente. En este mensaje, usa la numeración propia del lenguaje natural (que empieza en 1, no en 0)
8. Crea una lista llamada *linea_de_cajas* que contenga todas las cajas (las listas de los apartados anteriores). Será una lista cuyos elementos serán a su vez, listas.
9. A partir de linea_de_cajas, empleando bucles, muestra un informe similar a este:

   ```
   Caja con 2 clientes: Ernesto, Luis
   Caja con 3 clientes: Ana, Pedro, María
   Caje con 2 clientes: Marta, Juan
   ```

10. Modifica linea linea_de_cajas, para que pase a ser una lista de cadenas (no una lista de listas). Cada elemento será una única cadena, formada por los clientes de la caja, separados por el caracter punto y coma.

Práctica 1.2. Introducción a los diccionarios

Escribe el *script* `~/st/practica01/prueba_diccionarios.py`, que

1. Defina un diccionario donde la clave sea un nombre de ciudad o provincia, y el valor, una lista de comidas o bebidas típicas de ese lugar.
2. Añada nuevos elementos al diccionario.
3. Añada nuevos elementos a alguna de las listas
4. Defina la lista `alcoholicas` con elementos como `vino, sidra, cerveza`. Borre del diccionario todas las bebidas incluidas en esta lista.
5. Defina la lista `comunidad_valenciana` con los elementos `valencia, alicante, castellon`. En el caso de que en el diccionario esté alguna de estas provincias, le añade como plato típico `paella`
6. Borre todas las entradas con claves incluidas en `comunidad_valenciana`
7. Defina la función `muestra_alfabetico` que muestre el contenido del diccionario en un formato apropiado para que lo lea un humano (esto es, bien tabulado, sin corchetes, etc), por orden alfabético de las claves
8. Defina la función `muestra_por_longitud` que muestre el contenido del diccionario en un formato apropiado para que lo lea un humano, ordenado por el número de elementos de las listas

Práctica 1.3. Introducción a las funciones

Escribe el script `~/st/practica01/redondea_nota.py`, donde declararás y probarás la función `redondea_nota`. Esta función recibirá dos parámetros: `nota` y `modo`. Devolverá una cadena.

- Nota será la nota numérica de un estudiante, calificada de 0 a 10.
- La cadena devuelta será la calificación en forma de cadena (suspenso, aprobado, matrícula de honor... emplea las categorías que quieras).
- El parámetro `modo` podrá contener las cadenas `normal`, `laxo` o `estricto`. Según el modo, los criterios de calificación serán distintos. Por ejemplo, en modo estricto, una nota de 4.99 sería un suspenso. En modo laxo, una nota de 4 podría ser un aprobado.
- Aplica los criterios que consideres oportunos, de forma que la correspondencia entre la nota numérica y la calificación final sea más o menos estricta, según el parámetro modo.

Práctica 1.4 Ordenación

En esta práctica escribirás un script llamado `~/st/practica01/ordena.py`, donde probarás cómo ordenar una lista con el método sort(), definiendo una función para personalizar el órden.

- La estructura de datos a ordenar será una lista de listas, por ejemplo

  ```
  [ [1,3], [5], [0], [2,2,2] ]
  ```

  Una vez ordenada, su estado será

  ```
  [ [0, [1,3], [5], [2,2,2] ]
  ```

- Si hubiera listas que suman igual, primero debe aparecer la que tenga menor número de elementos
- Si alguna lista tuviera un elemento no permitido, esto es, algo diferente a un número, el script morirá (Usa `raise SystemExit`). La lista vacía es un elemento no permitido.
- Aunque esto es una prueba, hazlo razonablemente *bien*. Define las funciones que necesites. No uses más de 2 o 3 sentencias if anidadas.

Práctica 1.5 Cambia_nombre.py

En esta práctica escribirás un script llamado `~/st/practica01/cambia_nombre.py`, que, en el caso de que sea posible, modificará los nombres de todos los ficheros y directorios contenidos en ciertos directorios, para usar caracteres que sabemos que no causarán ningún problema en ningún sistema de ficheros: minúsculas, números, guiones, barras bajas y puntos.

1. Recibirá como argumento uno o más directorios. Mostrará un error por stderr y morirá si alguno de los argumentos no es un directorio. En caso de no recibir ningún argumento, se entiende que trabaja sobre el directorio actual
2. El script intentará cambiar el nombre a todos los ficheros y directorios del directorio o directorios indicados
3. Solo trabajará con el directorio actual, no recorrerá directorios y subdirectorios recursivamente
4. El cambio de nombre que intentará hacer será:
   - Reemplazar los espacios por barras bajas (_)
   - Cambiar todas las mayúsculas por minúsculas
   - Reemplazando la eñe por `nn` , `ny` , `gn` o alguna combinación similar
   - Reemplazando las vocales con tilde por la misma vocal sin tilde
   - Reemplazar cualquier carácter *raro*, como `|@#~!"·$%&` etc por un punto o una barra baja

Decimos *intentar* porque tal vez este cambio sin más no sea posible. Por ejemplo, un directorio que contenga los ficheros `HOLA!`, `hola!` y `hola.`

(Son tres nombres legales, pero al intentar cambiarlo, los tres generarían el mismo nombre simplificado. A esta situación la denominaremos *colisión*)

Práctica básica: Cuando se de un caso como este, el script indicará por salida estándar que ciertos ficheros no se modifican. Se limitará a cambiar lo que sea posible cambiar.

Prática optativa: En caso de colisión, el script buscará diferentes nombres para todos los ficheros. Por ejemplo añadiendo `.001`, `.002`, etc, entre el nombre y la extensión (si existe)

Busca en el módulo os.path la función que te indica si un fichero es un directorio. Para sustituir caractes en una cadena, puedes reconstruir una cadena carácter a carácter o usar métodos de los objetos string.

Práctica 1.6 optparse / argparse

En este ejercicio mejorarás el script del apartado anterior, usando la librería `optparse` o la librería `argparse`. Cualquiera de las dos es válida pero argparse es preferible, es más moderna y su uso algo más sencillo.

1. Una vez que hayas acabado el script anterior, hazle una copia llamada `~/st/practica01/cambia_nombre0.py`

   Ahora añadirás funcionalidad a

   `~/st/practica01/cambia_nombre.py`

   Observa que cambia_nombre0.py será el script especificado en la práctica 1.5 y `cambia_nombre.py`, el script de esta práctica.

2. Usando `optparse` o `argparse` , haz que

   `~/st/practica01/cambia_nombre.py` acepte los siguientes flags:

```
-h --help          Mostrar un mensaje de ayuda
-r --recursive     Recorrer los directorios recursivamente
-s --spaces        Reemplazar los espacios por barra baja
-c --case          Reemplazar las mayúsculas por minúsculas
-n --enne          Reemplazar la eñe por otra combinación de letras
-t --accent        Reemplazar las vocales con acento
-w --weird         Reemplazar los caracteres "raros"
```

Si el script no recibe ni -s ni -c ni -n ni -t ni -w, entonces se comportará como si hubiera recibido las opciones -s, -c, -n, -t, -w. Esto es, se comportará como `cambia_nombre0.py`

Práctica 1.7. Ficheros

En esta práctica mejorarás el script ordena.py, añadiéndole lectura y escritura de ficheros. Haz una copia de

`~/st/practica01/ordena.py`

ponle como nombre

`~/st/practica01/ordena0.py`

y ahora añadirás funcionalidad a

`~/st/practica01/ordena.py`

Observa que ordena0.py será el script especificado en la práctica 1.4 y ordena.py, el script de esta práctica.

Este script recibirá un fichero de texto con la misma estructura de datos que ordena0.py, pero representado en forma de texto, de forma que la lista de listas será un fichero de líneas, con elementos separados por comas.

Por ejemplo, la estructura

```
[ [1,3], [5], [0], [2,2,2] ]
```

se representaría así

```
1,3
5
0
2,2,2
```

De la misma forma, una vez ordenada la estructura, se representaría así

```
0
1,3
5
2,2,2
```

- El script leerá el texto desde el fichero indicado con la opción `-i --input`. El valor por omisión será la entrada estándar. Esto es, si no se indica ningún fichero, leerá desde stdin.
- El script escribirá el texto en el fichero indicado con la opción `-o --output`. El valor por omisión será la salida estándar. Esto es, si no se indica ningún fichero, escribirá en stdout
- En caso de que el script no pueda leer o escribir, deberá capturar la excepción correspondiente y mostrar un mensaje personalizado por la salida de error.
- La opción `-h --help` debe mostrar un mensaje de ayuda
- Si el fichero de entrada contiene alguna línea en blanco, en cualquier lugar, será válido.
- Si hay espacios o tabuladores antes o después de las comas, también será válido
- Si una línea acaba en coma, será incorrecta
- Si alguna línea contiene un elemento que no sea un número natural (incluyendo el cero), será incorrecta
- Si alguna línea no cumple con la especificación, el script debe mostrar el nº de línea incorrecto (contando las líneas desde 1), mostrar el texto de la línea incorrecta y terminar