
python


python

1. Introducción

En esta práctica se propone el desarrollo de una aplicación que agrupe algunas de las funcionalidades básicas que se podrían encontrar en un sistema de control de presencia en recintos de la Universidad de Murcia. Aunque en un sistema real estas funcionalidades se repartirían entre varias aplicaciones (una aplicación en el teléfono móvil del usuario que va a registrar su presencia, un servidor del sistema, etc.), vamos a agruparlas todas en un mismo programa a modo de prueba de concepto. El programa será capaz de:

1. Analizar un fichero textual con los logs del sistema de control de presencia.
2. Añadir nuevos logs al fichero para registrar la presencia de un usuario en cierto lugar mediante el escaneo de un código QR que identifica una localización de la Universidad de Murcia.
3. Filtrar y corregir errores en el fichero de logs, generando una salida con un formato distinto al de entrada.
4. Encontrar los usuarios que han coincidido con un usuario dado en la misma localización en una fecha determinada.

Partiendo del escenario propuesto, el objetivo de la práctica es trabajar los siguientes aspectos:

1. Validar y extraer información con expresiones regulares.
2. Realizar sustituciones para, partiendo de una entrada, generar una salida con otro formato.
3. Realizar cálculos eficientes con la información representada en estructuras de Python.

2. Formato del log de entrada

En el fichero `log.txt` que acompaña a este enunciado tenemos un ejemplo de *log* de registros de presencia del sistema que vamos a desarrollar. Puede contener dos tipos de líneas:

- Líneas con comentarios que comienzan por //
- Líneas que contienen registros de presencia (uno por línea).

Los registros de presencia contienen información sobre un instante de tiempo (día y hora), una localización (código patrimonial de la Universidad de Murcia), un identificador de usuario (número de móvil) y una acción que puede ser `in` o `out`.

Por ejemplo, en el fichero `log.txt` el primer registro de presencia es (se marcan en azul los espacios en blanco obligatorios):

```
[␣Day␣2020␣-␣11␣-␣02␣,␣09␣:␣23␣:␣26␣]␣Location␣:␣02.ED085.B1.0.012␣,␣User␣:␣687␣654␣321␣in
```

y esto indica que el usuario con teléfono móvil 687654321 accedió a las 09:23:26 horas del día 02/11/2020 al recinto 02.ED085.B1.0.012 (código patrimonial que identifica el aula A.03 del Aulario Norte). El formato es igual para el resto de líneas, variando únicamente los valores de los campos de fecha y hora, localización, identificador de usuario y acción.

Se considerará durante toda la práctica que los registros del fichero `log.txt` están ordenados cronológicamente.

2.1. Códigos patrimoniales

Los códigos patrimoniales de la Universidad de Murcia permiten identificar de manera unívoca cualquier recinto (aula, laboratorio, despacho de profesor, etc.). El inventario de bienes de la Universidad de Murcia emplea estos códigos para localizar dónde se encuentra el mobiliario, los equipos informáticos, etc. En un sistema de control de presencia se pueden emplear estos mismos códigos para identificar en qué aula o laboratorio se ha producido una entrada o salida.

En la dirección `https://www.um.es/web/universidad/mapas` se puede encontrar un plano de cada planta de cada edificio de la Universidad.

El formato de los códigos patrimoniales es el siguiente (se usa como ejemplo el aula 1.01 de la Facultad de Informática):

Cod.Campus	Cod.Edificio	Cod.Bloque	Cod.Planta	Cod.Sala
02	ED094	B1	1	079

- Cod.Campus: código del campus (opcional).
  - 01 : Campus de la Merced.
  - 02 : Campus de Espinardo.
  - 03 : Campus de Cartagena.
  - 04 : Campus de Ciencias de la Salud.
  - 05 : Campus de San Javier.
  - 06 : Campus de Lorca.
- Cod.Edificio: identificador del edificio, con el formato `EDXXX` o `ED_XXX`, donde `XXX` es un código numérico de tres dígitos. Este código numérico es único, es decir, no hay dos edificios en campus distintos con el mismo código.
- Cod.Bloque: identificador de un bloque del edificio, con el formato `BX` donde `X` es un código numérico de un dígito.
- Cod.Planta: identificador de la planta. Puede ser un entero negativo (sótano) o positivo (0=planta baja, y restantes pisos en adelante). También puede tomar el valor `EN` si es una entreplanta o `CU` si es una zona de cubierta (tejado).
- Cod.Sala: identificador de la sala, con el formato numérico `XXX`, que opcionalmente puede ir seguido de una cadena `.Y`, donde `Y` es una letra mayúscula o un número. Esta cadena opcional representa las posibles divisiones de una sala en subespacios.

Los campos del código patrimonial pueden separarse con el carácter guión bajo (`_`) o el carácter punto (`.`). Las siguientes líneas son representaciones válidas del código patrimonial del aula 1.01 de la Facultad de Informática (no hay espacios entre los caracteres):

```
02.ED094.B1.1.079
02_ED_094.B1.1.079
ED_094_B1.1.079
ED_094.B1_1_079
```

3. Inicio del Programa

Al iniciarse el programa, se mostrará un menú textual con las opciones disponibles. Cada opción se identificará por una letra. El usuario puede elegir una opción indicando la letra correspondiente (en mayúscula o minúscula). Si se introduce una opción incorrecta, el programa debe informar de ello, volver a mostrar el menú y solicitar la opción.

El menú deberá mostrarse con las siguientes opciones:

```
A: Añadir registro de presencia
G: Generar salida
C: Control de presencia
S: Salir del programa
>> _
```

En las siguientes secciones se describe la funcionalidad de las tres primeras opciones. Después de ejecutar una de ellas se volverá a mostrar este menú. No se considera necesario explicar cómo se tiene que implementar la cuarta opción.

4. Añadir registro de presencia

Seleccionando la opción A del menú principal, el programa solicitará los datos para incorporar una nueva línea al archivo `log.txt`. Este archivo se encontrará en la misma carpeta que el programa. Si no existe, se creará el archivo vacío. Los datos del nuevo registro de presencia se solicitarán en el siguiente orden:

1. Fecha (formato DD/MM/AAAA o bien DD-MM-AAAA).
2. Hora (formato HH:MM:SS).
3. Ruta de un fichero `.png` con el código QR del recinto.
4. Teléfono móvil del usuario (formato XXX XXX XXX con los espacios opcionales).
5. Acción (`in` o `out`) en cualquier combinación de mayúsculas o minúsculas.

Usando el paquete `datetime` será necesario comprobar que la fecha hace referencia a un día entre lunes y viernes, y que la hora está comprendida entre las 08:30:00 y las 21:00:00. Además, para mantener el orden cronológico, la fecha y hora deben ser posteriores a las del último registro del archivo `log.txt`. En la sección 10.1 se puede encontrar documentación sobre el paquete `datetime`.

En el caso del fichero `.png`, el programa debe:

1. Verificar que la ruta indicada termina en `.png` y que el fichero existe.
2. Decodificar el código QR y comprobar que contiene un código patrimonial válido.

Si todo es correcto, el programa debe insertar al final de `log.txt` una nueva línea con el registro, y también debe imprimirlo por consola. En caso contrario, si se produce un error al introducir un dato, el programa debe indicar que se trata de un valor erróneo y volverá a solicitarlo (los datos de campos correctos no se vuelven a pedir).

En caso de que se introduzca una cadena vacía, se considerará que el usuario desiste de añadir el nuevo registro, y el programa volverá a mostrar el menú de inicio.

El siguiente ejemplo corresponde a una operación de añadido correcta:

```
>> A
Fecha: 10/11/2020
Hora: 10:00:00
Ruta QR: ./recintos/Informatica_Laboratorio_1_01.png
Teléfono: 600111222
Acción: in
--
Registro añadido correctamente:
[ Day 2020 - 11 - 10 , 10 : 00 : 00 ] Location : 02.ED094.B1.1.079 , User : 600 111 222 in
```

Junto a este enunciado, en el Aula Virtual se puede encontrar el archivo `recintos.zip` que contiene varios códigos QR (en imágenes PNG) de diversos lugares del Campus de Espinardo. Atención, porque algunos de estos archivos contienen cadenas que no son códigos patrimoniales correctos. El programa debe estar preparado para detectar el error. En la sección 10.2 se indica el modo de decodificar códigos QR en Python a partir de imágenes PNG.

4.1. Listado de edificios

Para realizar la verificación de la cadena extraída del fichero `.png`, además de comprobar que cumple el formato especificado anteriormente, se debe emplear el archivo `edificios.csv` que contiene un listado de los edificios de la Universidad de Murcia, indicando por cada uno de ellos el código de campus en el que se encuentra, el código de edificio y su denominación:

```
Campus;Edificio;Nombre
1;1;Clínica Odontológica
1;2;Edificio Rector Sabater
1;3;Edificio Saavedra Fajardo
1;5;Rectorado
1;9;Colegio Mayor Azarbe
1;10;Facultad de Letras
1;11;Facultad de Derecho
1;12;Aulario de la Merced
...
```

Obsérvese que el código de edificio no contiene los dígitos 0 no significativos.

Con este archivo se puede comprobar que el código QR indica correctamente el código de campus y el del edificio. Además, en caso de que el código QR no contenga el código de campus, a partir de la información de este archivo es posible obtenerlo para completar el código patrimonial y generar el registro de log correctamente.

5. Generar salida

Mediante la segunda opción del menú principal del programa se generará un archivo de salida con el nombre `log.csv` a partir de la información que contenga en dicho momento el archivo de entrada `log.txt`.

5.1. Formato del log de salida

En muchos sistemas de minería de datos es necesario realizar una etapa previa al tratamiendo de la información que consiste en la adaptación del formato de entrada de los datos procedentes de alguna fuente, como puede ser un fichero de logs, al formato adecuado al sistema de minería de datos. Con objeto de facilitar este procesamiento, hay que convertir el contenido del fichero para filtrar los datos relevantes y dejar el fichero sin información superflua para esta aplicación.

En el caso de esta práctica, va a ser necesario generar un fichero de salida `log.csv` cuyas líneas de registros, resultado de la conversión del fichero de entrada `log.txt`, deben tener el siguiente formato:

```
Edificio;Bloque;Planta;Sala;Acción;Usuario;hh:mm:ss;dd/mm/aaaa
```

Los cuatro primeros campos se extraen del código patrimonial del recinto de la Universidad; el campo `Acción` contiene el indicador que aparece al final del registro (`in/out`), después viene el identificador del usuario (teléfono móvil), luego la hora y finalmente la fecha, todo ello separado por puntos y comas sin espacios. Debe respetarse este formato porque la corrección se hará por comparación de ficheros.

5.2. Registros erróneos

Puede suceder que el fichero de entrada contenga registros que no verifiquen el formato especificado en la sección 2. Cuando se detecte un registro incorrecto se debe indicar la línea que ocupa y no se debe volcar su información a la salida. También se debe comprobar que los códigos patrimoniales son correctos, usando la información de `edificios.csv`.

5.3. Registros redundantes

El fichero de entrada `log.txt` puede tener registros que llamamos redundantes. Dos registros son redundantes si cumplen todas las condiciones siguientes:

- Hacen referencia al mismo usuario (teléfono móvil).
- Entre ambos registros, no hay otro que haga referencia al mismo usuario.
- Hacen referencia al mismo recinto.
- Tienen el mismo valor de acción.

Si los registros redundantes tienen la acción `in`, se volcará a la salida el primero y se descartará el segundo. Si tienen la acción `out`, se volcará a la salida el segundo y se descartará el primero.

5.4. Registros faltantes

Un registro faltante es aquel que no permite cerrar correctamente un periodo de presencia de un usuario dentro de un recinto. Un periodo de presencia está cerrado correctamente en el fichero `log.txt` si tras el registro con la acción de tipo `in` de un usuario en un determinado recinto en un determinado día, existe un registro con una acción de tipo `out` del mismo usuario en el mismo

recinto en el mismo día, y no hay otro registro del mismo usuario entre ambos (la eliminación de registros redundantes se ha realizado previamente).

Hay varias formas de detectar registros faltantes:

1. Cuando el primer registro de un determinado usuario en cierto día tiene la acción `out`. En este caso, se inserta un registro `in` a las 08:30:00 de dicho día con el mismo usuario y recinto.
2. Cuando el último registro de un determinado usuario en cierto día tiene la acción `in`. En este caso, se inserta un registro `out` a las 21:00:00 de dicho día con el mismo usuario y recinto.
3. Cuando hay dos registros consecutivos para el mismo usuario con acción `in` (I1 e I2) en recintos distintos el mismo día. En este caso, se inserta un registro `out` antes de I2 para el mismo usuario a la hora indicada en I2 en el recinto de I1.
4. Cuando hay dos registros consecutivos para el mismo usuario con acción `out` (O1 y O2) en recintos distintos el mismo día. En este caso, se inserta un registro `in` después de O1 para el mismo usuario a la hora indicada en O1 en el recinto de O2.

5.5. Resumen de la salida

Al finalizar la generación del archivo `log.csv`, el programa debe mostrar por consola el número de registros válidos procesados, número de registros erróneos, número de redundantes y número de faltantes.

```
>> G
Generando fichero log.csv
--
Registros válidos: 629
Registros erróneos: 5
Registros redundantes: 2
Registros faltantes: 4
```

6. Control de presencia

La tercera opción del menú principal permitirá realizar un control de presencia. Para llevar a cabo la funcionalidad de esta opción, el programa empleará el fichero generado con la opción anterior, es decir `log.csv`. Teniendo en cuenta que los datos de este archivo están verificados, se podrá realizar una extracción de la información de las líneas del fichero `csv` con expresiones regulares sencillas. La información quedará guardada en memoria con las estructuras de datos que se consideren más adecuadas para implementar la funcionalidad de control de presencia de la forma más eficiente posible. Si el fichero `log.csv` no estuviese creado en el directorio del programa, se indicará al usuario que debe ejecutar la opción G del menú previamente.

Para realizar el control de presencia se solicitará el teléfono móvil de un usuario y una fecha, con un formato similar al usado en la opción para añadir registros (ver sección 4). El sistema generará por consola un listado de los recintos en los que estuvo presente el usuario en la fecha especificada y los teléfonos móviles de los otros usuarios que coincidieron con él en algún intervalo de tiempo superior a 5 minutos.

```
>> C
Teléfono: 662663664
Fecha: 10/11/2020
--
Aulario Norte ED085.B1.0.012
- 656646636
- 684692601
- 620194857
- 699887766
- 610203040
- 675849302
...
Aulario Norte ED085.B1.0.007
- 675849302
- 620194857
- 699887766
- 610203040
- 684692601
- 609906806
...
```

Si no se encontró ningún registro de presencia del usuario con el teléfono indicado y en la fecha introducida, se indicará la ausencia de información:

```
>> C
Teléfono: 656646636
Fecha: 06/11/2020
--
No se han encontrado registros de presencia.
```

7. Entrega

La entrega de la práctica se realizará mediante una tarea a la que habrá que subir el fichero Python con la implementación (`practica2.py`). En caso de que se haya implementado con varios ficheros, se deberán comprimir en un archivo zip (`practica2.zip`). Teniendo en cuenta que no se solicita una memoria de prácticas, y en deferencia a los profesores que harán la corrección, el código deberá estar convenientemente formateado y comentado de modo que se facilite su lectura y comprensión.

8. Puntuación

La práctica puede implementarse parcialmente. La puntuación de cada apartado (en caso de que se implemente correctamente) es la siguiente:

- Añadir registro de presencia: hasta 3 puntos.

- Generar salida:
  - Sin implementar control de registros erróneos, redundantes o faltantes: hasta 2
  - Eliminando registros erróneos: hasta +1
  - Eliminando registros redundantes: hasta +1
  - Añadiendo registros faltantes: hasta +1
- Control de presencia: hasta 2

Se penalizará la compilación repetida de la misma expresión regular.

En el total de la parte práctica de la asignatura, esta práctica puntúa el 60 %.

La puntuación de la práctica está sujeta a una posible entrevista al final del cuatrimestre.

9. Restricciones

Para centrar el desarrollo de la práctica en el uso de expresiones regulares, queda prohibido emplear métodos de cadenas de caracteres de Python, como `split()` o la comparación de cadenas. Únicamente se pueden emplear los operadores de concatenación (+), troceado (`[x:y]` o `[x:y:z]`) y las expresiones de formateo de cadenas (`%`) en caso de que no sea posible realizar la misma operación con sustituciones basadas en expresiones regulares. Si se tiene alguna duda en relación con estas restricciones, es mejor consultarla con la profesora o profesor de prácticas.

Todos los archivos manejados en esta práctica deben usar codificación `utf8`.

10. Información auxiliar

10.1. Uso del paquete `datetime`

El manejo de fechas y horas de esta práctica se puede llevar a cabo usando el paquete `datetime` de Python, que forma parte de la distribución estándar (no es necesario instalarlo con pip).

El tipo de dato adecuado para representar un instante de tiempo (fecha y hora) es `datetime`:

```
from datetime import datetime
# 07/11/2020 20:14:00
d = datetime(2020,11,7,20,14,0)
```

La creación del instante de tiempo se realiza indicando los argumentos en el orden: año, mes, día, hora, minuto, segundo. Se puede recuperar por separado la información de cualquiera de estos argumentos accediendo a los campos `year`, `month`, `day`, `hour`, `minute` y `second`:

```
d = datetime(2020,11,7,20,14,0)
print(d.year) # Imprime 2020
```

Si los datos usados para crear el `datetime` no son correctos (por ejemplo, porque se supera el número máximo de días del mes), se generará una excepción de tipo `ValueError`. Esta excepción,

junto con un uso previo de expresiones regulares, se puede emplear para verificar la validez de los datos.

El tipo de dato `datetime` permite usar los operadores relacionales (<, <=, ==, => y >) para comparar fechas en orden cronológico:

```
d1 = datetime(2020,11,7,20,14,0)
d2 = datetime(2020,11,7,21,0,0)
print(d1 < d2) # Imprime True
```

Se puede calcular el número de segundos que separa a dos instantes de tiempo ya que el tipo de dato `datetime` permite realizar operaciones aritméticas:

```
from datetime import datetime,timedelta
d1 = datetime(2020,11,7,20,14,0)
d2 = datetime(2020,11,7,21,0,0)
intervalo = d2-d1
print(intervalo.total_seconds())
```

Un intervalo entre dos instantes de tipo se representa con el tipo de dato `timedelta`. Este tipo de dato dispone del método `total_seconds()` para obtener en número de segundos del intervalo como un valor `float`.

Por otra parte, es posible comprobar el día de la semana de cualquier fecha usando el método `timetuple()` de `datetime`. Este método devuelve una tupla cuya entrada número 6 corresponde al día de la semana especificado como un entero entre 0 (lunes) y 6 (domingo).

```
d = datetime(2020,11,7,20,14,0)
tt = d.timetuple()
print(tt[6]) # Imprime 5 (sábado)
```

Si fuese necesario realizar comparaciones únicamente con horas (sin especificar día), se podría usar el tipo de dato `time` del mismo paquete:

```
from datetime import time
h1 = time(8,30,0)
h2 = time(9,30,0)
h3 = time(21,0,0)
print(h1 < h2 < h3) # Imprime True
```

10.2. Decodificación de códigos QR

Los códigos QR (*Quick Response* code) fueron creados en 1994 por la compañía japonesa Denso Wave, subsidiaria de Toyota, con el objetivo de codificar información para ser extraída a alta velocidad (de ahí su nombre). La compañía los utilizó para hacer el tracking de los coches durante el proceso de fabricación. Se convirtieron en un estándar ISO/IEC en 2000 (fuente `https://www.qrcode.com/`).

Un código QR se estructura en una matriz bidimensional con celdas de dos colores contrastados, usualmente blanco y negro (se inspiraron en el tablero de Go para crearlos). Hay varias versiones de códigos QR según la cantidad de celdas que forman la matriz: van desde la versión 1 (con una matriz de 21 x 21 celdas) hasta la versión 10 (con 177 x 177 celdas). Las versiones de más

celdas consiguen almacenar mayor cantidad de información. Los códigos más extendidos para el uso del público en general suelen ser los de 25 x 25 y de 29 x 29, para captura desde el teléfono móvil en cualquier situación (paquetes de productos, folletos de mano, tarjetas o carteles de pared). Por ejemplo, la Universidad de Murcia emplea códigos de 25 x 25 para la codificar el código patrimonial de los recintos.

Para decodificar códigos QR en Python necesitamos instalar el paquete `opencv-python`. Como sabemos, para instalar paquetes tenemos que abrir una consola y usar el comando pip (o pip3):

```
pip install opencv-python
```

Con el paquete ya instalado, podemos usar el siguiente código Python para decodificar una imagen `.png` con el código patrimonial:

```
1  import cv2
2  imagen = cv2.imread('./recintos/Informatica_Salon_Grados.png')
3
4  if imagen is None:
5      print('Error leyendo fichero')
6  else:
7      detectorQR = cv2.QRCodeDetector()
8      texto, puntos, _ = detectorQR.detectAndDecode(imagen)
9
10     if puntos is not None:
11         print(texto)
12         cv2.imshow('Codigo QR', imagen)
13         cv2.waitKey(0)
14         cv2.destroyAllWindows()
15     else:
16         print('Código QR no detectado')
```

La detección en sí se realiza en la línea 8. Si el segundo valor de la tupla devuelta por el método `detectAndDecode` no es nulo, la decodificación ha funcionado y se puede emplear el texto obtenido. En las líneas 11 a 14 se muestra el texto por consola y se visualiza la imagen en una ventana.

10.3. Codificación de códigos QR

Existen numerosos generadores de códigos QR gratuitos en Internet con los que puedes preparar nuevos ficheros `.png` para realizar pruebas. Por ejemplo:

`http://qr.calm9.com/en/`

Aunque no es necesario integrar en el código Python de esta práctica la generación de ficheros `.png` con códigos QR, quizás te interese saber cómo se puede usar alguna librería para esto.

Hay bastantes paquetes disponibles para producir códigos QR. Aquí vamos a centrarnos en `qrcode`, que se puede instalar fácilmente con el comando pip (o pip3). La documentación de la librería se puede consultar aquí:

`https://github.com/lincolnloop/python-qrcode`

A continuación se muestra un ejemplo completo que se comenta después.

```
1  import qrcode
2  qr = qrcode.QRCode(
3      version=2, # 25x25
4      error_correction=qrcode.constants.ERROR_CORRECT_M,
5      box_size=20,
6      border=2)
7  qr.add_data('Cadena que se almacena en el código')
8  img = qr.make_image(fill_color="black", back_color="white")
9  archivo_qr = open('fichero.png', 'wb')
10 img.save(archivo_qr)
11 archivo_qr.close()
```

En las líneas 2 a 6 se configura la librería para generar un código QR con determinadas características. El parámetro `version` establece cuántas celdas contendrá el código (2=25x25). Mediante `error_correction` podemos establecer qué resistencia tendrá el código protector de errores que se usará para generar el gráfico QR. Con `ERROR_CORRECT_M` se establece una protección del 15 %, es decir, se pueden recuperar completamente los datos originales teniendo hasta un 75 % de los bytes (ya sean de datos o del código de protección de errores) bien recuperados. Con `box_size` se determina el tamaño en número de pixels que tendrá el lado de las celdas. Por último, `border` indica el tamaño del borde del código, expresado en número de celdas en blanco.

La línea 7 añade los datos de la cadena que se va a codificar dentro del código QR. Las líneas 8 a 11 genera el archivo PNG con el código, usando el negro las celdas marcadas y el blanco para el fondo.