Proyecto de Prácticas
Autómatas y Lenguajes Formales
Curso: 2024/25

Convocatoria ordinaria de enero

Autores:

Nombre completo	Subgrupo	DNI
Guillén García, Rafael	2.4	

ÍNDICE

1. Descripción de la aplicación y Manual de usuario

La aplicación que se propone desarrollar como parte del proyecto de programación consiste en un analizador de ficheros de registros de eventos. En concreto, registros de compras realizadas desde un teléfono móvil. La aplicación se debe poder manejar a través de la línea de comandos pasándole valores al invocarla.
La sintaxis para usar la aplicación será exactamente la siguiente:
python main.py -n <fichero> | -sphone <telefono> <fichero> | -snif <NIF> <fichero> |-stime <desde> <hasta> <fichero> | -slocation <desde> <hasta> <fichero>

La opción -n llevará a cabo la normalización del fichero indicado en el siguiente argumento. Es
decir, mostrará por pantalla un listado equivalente, pero con todos los instantes temporales expresados con el primer formato indicado en el anexo 1 y todas las coordenadas geográficas
expresadas con el formato GPS indicado en el anexo 2.

La opción -sphone realizará un filtrado del fichero indicado como último argumento. El filtrado
consistirá en mostrar únicamente las entradas cuyo teléfono coincida con el indicado como segundo argumento.

La opción -snif realizará un filtrado del fichero indicado como último argumento. El filtrado consistirá en mostrar únicamente las entradas cuyo NIF coincida con el indicado como segundo
argumento.

La opción -stime realizará un filtrado del fichero indicado como último argumento. El filtrado consistirá en mostrar únicamente las entradas correspondientes a eventos producidos entre los dos instantes temporales indicados en los argumentos <desde> y <hasta>.

La opción -slocation no está implementada.

2. Formato de los datos (Modularización)

1. **Coordenadas:** las expresiones regulares usadas para filtrar formatos, y las funciones utilizadas son “sonCoordenadasCorrectas”, “convertir_a_GPS”, “convertirGPS”, “convertirSexagesimalADecimal”, “convertirGPSADecimal” y “convertirFormatoCoordenadas” se encargadas de comprobar si son correctas y de transformar todas las coordenadas al formato GPS.

2. **DNI:** Incluye las posibles letras que puede tener el NIF, la función “letraDNI” que calcula la letra que corresponde al NIF y la función “comprobarDNI” para comprobar si el NIF es correcto. La expresión regular comprueba que el patrón sea correcto y la definición de la función “snif”

3. **Fechas:** Incluye el diccionario “meses” que asigna a cada mes escrito su valor numérico, la función “bisiesto”, la función “esFechaCorrecta” y “esHoraCorrecta” que comprueban que la fecha introducida sea acertada, la función “convertirFormatoFecha” que cambia el formato de todas las fechas que cumplen con las expresiones regulares al formato 1 del anexo y la función “compararInstTemp” la cual es necesaria para “stime”

4. **Telefono:** Incluye la expresión regular para filtrar los teléfonos que entran y la definición de la opción “sphone”

5. **Normalizar:** Incluye una función para normalizar el instante temporal: “normalizarInstante” y otra para normalizar las coordenadas: “normalizarCoordenada”. Se define la opción -n

6. **Main:** Incluye las salidas que corresponden a las opciones “-n”, “-sphone”, “-snif” y “-stime”

3. Aspectos principales

El proyecto se basa en seis archivos fuente. Cinco de ellos son módulos que contienen las definiciones y el código necesario para implementar las funciones principales, asegurando su correcto funcionamiento.

En el módulo coordenadas.py, encontramos las definiciones de las expresiones regulares de las funciones necesarias para poder normalizar en el formato pedido las coordenadas que se encuentren en el fichero indicado como parámetro.

En el módulo dni.py, encontramos una lista de letras, la cual usaremos para saber qué letra corresponde a un patrón de números, y de esta forma saber si un NIF es correcto. Estas funciones las usaremos en la definición de la función de la opción “-snif”.

En el módulo fechas.py, encontramos un diccionario el cual asigna a cada mes su número correspondiente, encontramos funciones para verificar si una fecha es correcta (teniendo en cuenta los años bisiestos) y si una hora es correcta. También tenemos una función para comparar dos instantes temporales, teniendo en cuenta los años, meses, días, horas, minutos y segundos. Además, encontramos las expresiones regulares para cada uno de los formatos pedidos en el boletín. Estas expresiones son necesarias para la función convertirFormatoFecha, que, dada una cadena, comprueba si se corresponde a un instante temporal y además devuelve un diccionario indicando el año, mes, día, horas, minutos y segundos. Todas estas funciones son necesarias para la implementación de la función de la opción “-stime”.

En el módulo normalizar.py, encontramos la definición de las funciones normalizarInstante y normalizarCoordenadas, que con el uso de funciones definidas en los módulos fechas.py y coordenadas.py, respectivamente, normalizan el instante o coordenadas dados en el formato pedido. Con estas funciones, definimos la función de la opción “-n”.

En el módulo teléfono.py, encontramos una expresión regular para los números de teléfono de distintos países. Con esta expresión definimos la función de la opción “-sphone”.

Por último, encontramos el main.py, archivo fuente en el que, según la opción y los parámetros dados, llama a la función necesaria para responder a la petición del usuario.

El código de cada archivo fuente se explica mejor mediante comentarios dentro de dicho archivo.

El primer problema que resolvimos fue el “**sphone**” tomando la expresión regular para teléfonos solo de España “ \d{3} \d{3} \d{3} ” la cual se filtra con el fichero y devuelve si encuentra alguno que se corresponda con el patrón, más tarde añadimos a la expresión regular las características para que pudiera reconocer teléfonos de otros países además de España “ (\d\s?){9}|(?:\+(\d{1,3})\s?)?(\d\s?){9,13}$ ” en la que la primera parte reconoce los números de longitud 9 dando igual si están separados por algún lado o no, y la segunda parte se divide en dos, 1 que tenga el símbolo “+” seguido de uno a tres dígitos y, después los espacios que desee; y 2 reconoce dígitos dependiendo de lo anterior con un máximo de 15 dígitos y un mínimo de 10 separado en los grupos que se desee.

Después resolvimos “**snif**” en el cual usamos una expresión regular “(\d{8}[A-Z])|([X-Z])(\d{7}[A-Z])” la cual su primera parte filtra para DNI español y la segunda para DNI extranjero además de una función que calcula la letra que tiene el DNI, por último, se comprueba si hay alguno igual en el fichero o no.

El problema “**stime**” fue más complejo, en el teníamos que filtrar por tres tipos de patrones y normalizar para que todos estuvieran el mismo formato a la hora de compararlos con los parámetros de entrada y para imprimirlo haciendo uso de la función “compararInstTemp” y del archivo normalizar con la función “normalizarInstante”, el fragmento de código correspondiente es este:

```
if(compararInstTemp(instante_desde, evento) == -1
or compararInstTemp(instante_desde, evento) == 0)
and(compararInstTemp(evento, instante_hasta) == 1
or compararInstTemp(evento, instante_hasta) == 0):
        normalizar.normalizarInstante(evento)
```

Las expresiones ultizadas para cada formato han sido estas:

1. Formato 1: “(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2})” filtra la fecha en formato YYYY-MM-DD HH:MM
2. Formato 2:
   “(?i)(January|February|March|April|May|June|July|August|September|October|November|December)\s+(0?[1-9]|[12][0-9]|3[01]),\s+(\d{1,4})\s+(1[0-2]|0?[1-9]):([0-5][0-9])\s+(AM|PM)”
   Esta es más compleja ya que tiene como entrada una cadena de texto. Empieza con “?i” que hace que no discrimine entre mayúsculas y minúsculas, se le meten las posibles cadenas que pueden entrar, luego las posibles fechas que no son exactas pero serán comprobadas mas tarde en “compararInstTemp”, por ultimo (AM|PM) que será formateadas mas tarde en este fragmento de código:

```
 if am_pm.lower() == 'pm' and horas != '12':
     horas = str(int(horas) + 12)
 elif am_pm.lower() == 'am' and horas == '12':
     horas = '00'
```

3. Formato 3: “(\d{2}):(\d{2}):(\d{2}) (\d{2})/(\d{2})/(\d{4})” filtra la fecha en formato HH:MM:SS DD/MM/YYYY

También hemos realizado el apartado de normalizar las coordenadas las cuales tienen también tres formatos:

Formato 1: “([+-]?\d{1,2}(?:\.\d+)?)\s*,\s*([+-]?\d{1,2}(?:\.\d+)?)\s*” el cual reconoce coordenadas en formato decimal, entra o no el signo, luego dos dígitos un “.” y decimales, después espacios, una coma y repetir lo anterior

Formato 2: “(\d{1,2}°\d{1,2}\'\d{1,2}(?:\.\d{1,4})?"\s*[NS])\s*,
\s(\d{1,2}°\d{1,2}\'\d{1,2}(?:\.\d{1,4})?"\s*[EW])” reconoce las coordenadas en formato sexagesimal, primero coge la latitud, uno o dos dígitos para los grados, uno o dos para los minutos y uno o dos para los segundos enteros y cuatro para los decimales, después si es norte o sur para saber la orientación; y lo mismo para la longitud (este y oeste).

Formato 3: “(\d{3}\d{2}\d{2}\.\d{4}[NS])\s*,\s*(\d{3}\d{2}\d{2}\.\d{4}[EW])” reconoce las coordenadas en formato GPS, primero la latitud donde tiene 3 digitos para los grados, dos para los minutos y dos para los segundos enteros y cuatro para los decimales después si es norte o sur para saber la orientación; y lo mismo para la longitud (este y oeste).

Por último, el problema “-n”, del cual hemos hablado antes, este cuenta con dos funciones para transformar el formato de las fechas al formato “YYYY-MM-DD HH:MM” y de las coordenadas al formato GPS.

4. Conclusiones

Pues nos ha parecido un proyecto bastante interesante para aprender distintas funcionalidades de Python. Personalmente hemos descubierto un lenguaje algo más amigable y cómodo que los ya conocidos y la verdad, queremos seguir formándonos más de él, por otro lado, hemos tenido complicación a la hora de acostumbrarnos a la sintaxis de Python, ya que estamos acostumbrados a la de otros lenguajes, como C++, y alguna vez hemos tenido errores por confundir la sintaxis, y también al normalizar los formatos tanto de fecha como de coordenadas, pero, al final, lo hemos conseguido y hemos de reconocer que nos parece una sintaxis sencilla y con la que es más fácil trabajar.

También hemos intentado implementar estructuras que hemos aprendido por asignaturas como son las clases vistas en AED1 viendo la similitud entre los distintos lenguajes con Python, pero nos ha presentado problemas al ser algo que no tenemos dominado en Python y, por tanto, hemos decidido trabajar con listas y diccionarios, cosas con las que estamos más familiarizados.