Autómatas y Lenguajes Formales

Boletín de Prácticas

Curso 2024-2025

Grupos 1, 2, 3 y 9

TABLA DE CONTENIDOS

Proyecto de programación

Descripción del trabajo a realizar

Las prácticas de la asignatura Autómatas y lenguajes Formales se realizan por parejas y consisten en el desarrollo de un proyecto de programación que permita al estudiante aplicar los conocimientos sobre expresiones regulares adquiridos durante el curso. En concreto, **debe realizarse una aplicación que cumpla los siguientes requisitos**:

- Utilizar el lenguaje Python
- Utilizar expresiones regulares para realizar las siguientes acciones:
  - Comprobar la validez de formato
  - Realizar búsquedas y sustituciones
  - Extraer información (filtrado)
- Los datos a utilizar estarán en ficheros de texto o en páginas web.

A lo largo de las sesiones de prácticas guiadas se trabajarán todos los aspectos necesarios para desarrollar una aplicación que cumpla los anteriores requisitos. Se podrá entregar esta aplicación como proyecto, o utilizar los conceptos aprendidos para desarrollar una aplicación de libre elección, siempre que cumpla los mismos requisitos. En este último caso se debe contactar lo antes posible con el profesor para obtener su visto bueno.

Además del código se deberá escribir y entregar un informe describiendo el trabajo realizado, que constará de los siguientes apartados:

- **Portada** con la siguiente información: Asignatura, Apellidos, Nombre, DNI, Grupo y Subgrupo de Prácticas, Convocatoria, Año académico y Profesor.
- **Índice de contenidos**.
- **Descripción de la aplicación**: Objetivo y funcionamiento general de la aplicación.
- **Manual de usuario**: Instrucciones para utilizar la aplicación.
- **Formato de los datos**: Descripción del formato de todos los datos usados.
- **Aspectos principales**: Explicación de cómo has resuelto los distintos problemas usando expresiones regulares.
- **Conclusiones**: Comentarios personales con relación a las dificultades encontradas, la forma de superarlas y el grado de satisfacción con la asignatura, así como sugerencias de mejora.

El texto del informe se debe escribir usando una fuente similar a **Arial**. El **código fuente** incluido en el informe deberá estar bien formateado. Es decir, se debe cuidar el sangrado y usar un tamaño de la letra adecuado para que no se corten los renglones, o que cuando esto ocurra el código siga siendo legible y ordenado. El tipo de letra para **el código fuente** será `courier` o uno semejante de ancho fijo.

Entregables

Todo el material se entregará **comprimido en un único fichero cuyo nombre será una lista de los DNIs de sus autores separados por guiones (-) y cuyo contenido estará distribuido en carpetas** del siguiente modo:

- **Carpeta "*code*"**: Contendrá únicamente los ficheros siguientes:
    - Ficheros de código fuente.
    - Otros ficheros necesarios para compilar o ejecutar la aplicación tales como imágenes, textos, datos, etc.
- **Carpeta "*doc*"**: Contendrá el informe en formato PDF.

IMPORTANTE: El código debe compilar y ejecutar sin avisos (warnings) ni errores de compilación en unas condiciones similares a las de los laboratorios de prácticas.

Fecha y modo de entrega

La entrega se realizará a través de una tarea publicada en el aula virtual. Sólo debe hacer la entrega el miembro del grupo que vaya primero en la lista ordenada de alumnos de clase. La fecha límite para entregar el trabajo se indicará al publicarse la tarea y se avisará con antelación suficiente a través del aula virtual.

Criterios de evaluación

Si el profesor lo considerase oportuno se convocará a los estudiantes a la realización de una entrevista personal para aclarar detalles del trabajo realizado. En tal caso, la no asistencia (injustificada) a esta entrevista de prácticas será motivo de suspenso.

El proyecto de programación será valorado por el profesor con una puntuación de 0 a 10. Para superar la parte práctica de la asignatura será necesario obtener una nota igual o superior a 5 sobre 10.

En la evaluación se tendrá en cuenta la presentación, claridad y completitud del informe, el uso de un lenguaje técnico apropiado, la legibilidad, organización y documentación del código fuente, y la corrección, robustez, modularidad, extensibilidad, reusabilidad y eficiencia del software.

A continuación, se muestran algunos ejemplos de aspectos relevantes que se tendrán en cuenta durante la evaluación del software:

1. Legibilidad
   - Elige los identificadores de variables y funciones de forma adecuada.
   - Documenta las funciones incluyendo precondiciones, efecto y resultados.
   - Formatea el código fuente aplicando sangrado.
2. Corrección
   - No se producen errores en tiempo de ejecución.
   - Asegura la corrección de los datos introducidos por el usuario.
   - No se producen bucles infinitos ni llamadas recursivas que desborden la pila.
3. Robustez
   - Da valor inicial a las variables.
   - Controla errores (excepciones) producidos en tiempo de ejecución.
   - Comprueba los valores devueltos por funciones que informen sobre errores.
4. Modularidad
   - Aplica abstracción procedimental creando funciones para no repetir código.
   - Distribuye el código en módulos de forma coherente cuando sea necesario.
5. Extensibilidad y Reusabilidad
   - Crea constantes en lugar de usar valores literales en el código.
   - Evita el uso de variables globales.
   - Parametriza las funciones y módulos para hacerlos más generales.
6. Eficiencia
   - Selecciona la representación más adecuada para las estructuras de datos.
   - Utiliza esquemas de inserción, acceso, recorrido y búsqueda eficientes y adecuados.
   - Evita la repetición innecesaria de cálculos o acciones previamente realizadas.

Propuesta de aplicación

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** se corresponde con una ampliación que se explica en la sección ¿Cómo conseguir la mejor nota posible?.

El programa debe ser robusto y tolerar errores de formato. Es decir, en caso de que el fichero analizado contenga entradas con errores de formato, el programa los ignorará y seguirá procesando, la siguiente entrada.

El formato de los ficheros de registros de eventos que debe analizar el programa se explica en la sesión 3.

Sesiones de prácticas guiadas

Sesión 1. Introducción a Python I

Objetivos: Manejar el editor Pycharm y conocer el uso de variables, condiciones, bucles while y la definición de funciones propias en python.

Tareas

1) Escribe una función que reciba como parámetro un año y devuelva verdadero si es bisiesto.
2) Escribe una función que reciba como parámetro tres números representando año, mes y día y devuelva verdadero si se trata de una fecha correcta teniendo en cuenta si el año es bisiesto o no y la cantidad de días que tiene cada mes.
3) Escribe una función que dados tres números representando horas, minutos y segundos devuelva verdadero si se trata de una hora correcta o no.
4) Escribe una función que compare dos instantes temporales (fecha y hora) y devuelva -1 si el primero es el más antiguo, cero si son iguales y 1 si el segundo es más antiguo. La función tendrá 12 parámetros, pero más adelante veremos cómo se puede reducir este número.
5) Escribe un programa que genere y muestre por la pantalla 100 instantes temporales aleatorios pero correctos.

Sesión 2. Introducción a Python II

Objetivos: Uso de cadenas y listas en Python.

Tareas

1) Escribe una función que determine la letra correspondiente a un número de DNI dado.
2) Escribe una función que determine si la cadena recibida como parámetro representa un NIF válido. Es decir, si incluye los dígitos y la letra que les corresponda.
3) Escribe una función que genere una lista con NIF aleatorios, correctos y no repetidos del tamaño que se le indique como parámetro. La función devolverá esta lista de cadenas.
4) Usa **sys.argv** para hacer que el programa principal determine, según los argumentos empleados en su invocación, lo que debe hacer. Organiza el código creando funciones para llevar a cabo las distintas opciones. Obviamente, de momento las funciones estarán vacías.

Sesión 3. Introducción a Python III

Objetivos: Entrada/Salida, funciones open, read, write y close; módulos y paquetes

Tareas:

1) Escribe una función que genere un listado de compras realizadas desde un teléfono móvil. El listado se guardará en un fichero de texto en formato CSV sin cabeceras y usando punto y coma como separador de campos. La información asociada a cada compra constará de los siguientes campos: número de teléfono desde el que se realizó la operación, NIF del comprador, el instante temporal en el que se realizó la operación, las coordenadas geográficas dónde estaba el móvil en el momento de la compra, el producto comprado y el precio pagado en euros. Tanto los instantes temporales, como las coordenadas geográficas deben estar expresados usando los diferentes formatos

explicados en los anexos de este boletín. Puedes buscar en internet catálogos y listados de productos para generar los tuyos o puedes inventar los nombres de los productos uniendo palabras de conjuntos seleccionados. Ejemplos:

666 777 999 ; 12345678Z ; 2000-02-09 22:30 ; 30.0, -40.5 ; iMac ; 2300€
566 677 899 ; 87654321X ; May 20, 2000 10:30 pm ; 30° 0‘ 0.0000" N, 40° 30‘ 0.0000" W ; Ipad ; 600€
673 723 229 ; 12345678Z; 22:30:00 05/03/2002 ; 0300000.0000N0403000.0000W ; Iphone ; 399.99€

El formato de los números de teléfono debe ser el siguiente:

- 9 dígitos agrupados de tres en tres con un espacio en blanco entre los grupos.

El formato de los NIF debe ser el siguiente:

- 8 dígitos seguidos de una de las 23 letras del alfabeto cuando se trate de un DNI.
- Las letras X, Y o Z seguidas de 7 dígitos y terminando en una de las 23 letras del alfabeto cuando se trate de un NIE.

El precio será un número entero o real con decimales seguido del símbolo del euro.

Recuerda generar algunos casos incorrectos para, llegado el momento, poder probar la robustez de tu programa. También es buena idea que pruebes con ficheros generados por otros compañeros.

Sesión 4. Expresiones Regulares I

Objetivos: Manejar la sintaxis extendida de expresiones regulares

Tareas:

1) Escribe una expresión regular que permita validar el formato de un número teléfono.
2) Escribe una expresión regular que permita validar el formato de un NIF.
3) Escribe una expresión regular que permita validar el formato de un instante temporal.
4) Escribe una expresión regular que validar el formato de una coordenada geográfica.

Puedes hacer las pruebas que necesites en EditPad Pro y en la web https://regex101.com/

Sesión 5. Expresiones Regulares II

Objetivos: Uso de los modificadores, los grupos, los operadores look ahead y las sustituciones

Tareas:

1) Modifica las expresiones regulares de las sesiones anteriores para que, usando la captura de grupos, sea posible diferenciar cada campo capturado.

Sesión 6. Introducción a Python IV

Objetivos: Uso de expresiones regulares en Python, paquetes re y regex, compilación, matching y sustitución

Tareas:

1) Escribe una función que, dada una cadena de texto con un teléfono, determine si el formato es válido y, en caso afirmativo, extraiga el número.
2) Escribe una función que, dada una cadena de texto con un NIF, determine si el formato es válido y, en caso afirmativo, extraiga el número y la letra y compruebe si es un NIF correcto usando las funciones de las sesiones anteriores.

3) Escribe una función que, dada una cadena de texto con un instante temporal, determine si el formato es válido y, en caso afirmativo, extraiga el día, el mes y el año, la hora, los minutos y los segundos y compruebe si es una fecha y hora válidas usando las funciones de las sesiones anteriores.
4) Escribe una función que, dada una cadena de texto con una coordenada geográfica determine si el formato es válido y, en caso afirmativo, extraiga la longitud y latitud de la coordenada, y compruebe si se trata de un dato correcto.

Nota: Todas las expresiones regulares y las operaciones realizadas con las mismas deberán ser resueltas usando los paquetes re o regex.

Sesión 7. Introducción a Python V

Objetivos: Uso de diccionarios para representar datos estructurados

Tareas:

1) Modifica las funciones de los ejercicios de la sesión 6 para que devuelvan diccionarios representando las estructuras de datos obtenidas a partir de las cadenas de texto.
2) Escribe funciones que permitan mostrar por pantalla los diccionarios que representan los diferentes elementos obtenidos en las funciones anteriores. Ten en cuenta que te interesa poder mostrarlos usando diferentes formatos.
3) Escribe una función que permita validar el formato de una entrada del listado de compras. La función devolverá un diccionario que contenga los valores (diccionarios) de los distintos campos extraídos siempre y cuando todos sigan un formato correcto. Si algún campo de la entrada no es correcto te interesa devolver `None` para que el invocador de la función pueda detectar que la entrada era incorrecta.
4) Rellena las funciones que dejaste vacías en la sesión 2 para completar el programa.

¿Cómo conseguir la mejor nota posible?

Para aspirar a la máxima nota se debe escribir el código de la aplicación usando exclusivamente las construcciones del lenguaje y bibliotecas explicadas en clase o en los manuales existentes en el aula virtual. En especial, la resolución de los problemas relacionados con expresiones regulares deberá llevarse a cabo con los paquetes re o regex.

Haciendo el proyecto de programación propuesto en este boletín de prácticas, es decir, que incluya todo lo descrito en los guiones de las sesiones guiadas, se puede alcanzar una nota máxima de 7 puntos. Para aspirar a una calificación superior se deben añadir otras características. A continuación, se muestran algunos ejemplos junto con su descripción y valor. El máximo posible son 10 puntos. Si haces alguna de estas mejoras añade su hashtag en el texto del formulario al entregar la tarea a través del aula virtual:

- **#Telefonos (1 punto)**: Añade flexibilidad a la hora de expresar los números de teléfono. Consigue que los teléfonos puedan expresarse de forma simplificada con 9 dígitos, como en los ejemplos de este boletín, en cuyo caso se asumirá que son teléfonos españoles, pero permite que se puedan expresar con un formato similar al E.164. Concretamente, el formato de teléfono consistirá en un símbolo + seguido de diez a quince dígitos agrupados como se desee y con un espacio en blanco entre cada dos grupos. Ejemplos válidos serían +34 666 777 999, +1 123 6666 77 9999
- **#Distancias (2 puntos):** Añade al programa la posibilidad de mostrar todos los registros correspondientes a eventos producidos a una distancia máxima de cierta coordenada geográfica. Hazlo añadiendo la opción de filtrado -slocation <desde> <hasta> <fichero> que funcionará así:

  La opción -slocation realizará un filtrado del fichero indicado como último argumento. El filtrado consistirá en mostrar únicamente las entradas correspondientes a eventos producidos a una distancia en kilómetros inferior a la indicada como parámetro <hasta> tomando como origen para medir la distancia la coordenada indicada como parámetro <desde>. Para calcular la distancia entre dos coordenadas geográficas debes usar la fórmula de Haversine y el paquete math

Anexo 1. Tiempo

Existen muchos formatos distintos para expresar un instante en el tiempo. En este proyecto vamos a trabajar con los tres que se indican a continuación:

Formato 1

YYYY-MM-DD HH:MM

Primero aparece el año el mes y el día que se indican separados por guiones. Todos los números deben tener ceros para rellenar la cantidad máxima de dígitos que pueden representar. A continuación, tras uno o más espacios aparecen la hora y los minutos separados por el símbolo de los dos puntos. Los valores válidos para expresar los meses van desde 01 hasta 12 y para expresar los días desde 01 hasta 31. Los valores válidos para expresar las horas van desde 00 hasta 23 y para los minutos desde 00 hasta 59.

Ejemplo: `1945-08-06 08:15`

Formato 2

Month D, Y HH:MM AM/PM

Primero aparece el mes expresado mediante su nombre en inglés (mayúsculas o minúsculas están permitidas). A continuación, tras uno o más espacios aparece el número del día. Los valores válidos para el día van desde 1 hasta 31. Después, tras una coma y uno o más espacios, aparece el número que representa el año. Los años válidos van desde el 0 hasta el 9999. Después, tras uno o más espacios, aparecen la hora y los minutos separados por el símbolo de los dos puntos usando el sistema horario de 12 horas. Los minutos se rellenan con cero cuando se trata de valores inferiores a 10, pero las horas no. Los valores válidos para expresar las horas van desde 1 hasta 12 y para los minutos desde 00 hasta 59. Finalmente, tras uno o más espacios aparecen las palabras “AM” o “PM” (mayúsculas o minúsculas están permitidas), para indicar si se trata de horario de mañana o de tarde. Recuerda que 12:00 AM significa media noche y 12:00 PM significa medio día. Y que cinco minutos después de la media noche se escribe 12:05 AM, pero la una y media de la noche (01:30 en formato 24h) se escribe 1:30 AM.

Ejemplo: `August 6, 1945 8:15 AM`

Formato 3

HH:MM:SS DD/MM/YYYY

Primero aparece la hora expresada mediante tres grupos de dos dígitos (rellenando con cero cuando sea necesario) separados por el símbolo de los dos puntos. En este caso sí aparecen los segundos. Después, tras uno o más espacios aparecen tres grupos de dígitos separados por el símbolo de la barra de dividir. Primero el día, después el mes y, finalmente, el año. En este caso también se rellenan con cero los días o meses con valores inferiores a 10, y los años hasta completar cuatro dígitos. Y los valores válidos para días y meses también van desde 01 hasta 31 y desde 01 hasta 12.

Ejemplo: `08:15:00 06/08/1945`

Anexo 2. Coordenadas geográficas

Una coordenada geográfica consiste en una pareja de valores, el primero indica la latitud (distancia angular entre el punto y el ecuador) y el segundo la longitud (distancia angular entre el punto y el meridiano de Greenwich). Los valores permitidos para la latitud están entre -90.0° y 90.0°. Los valores permitidos para la longitud están entre -180.0° y 180.0°. Para expresar una coordenada geográfica se pueden usar distintos formatos. En este proyecto vamos a trabajar con los siguientes:

Decimal

Los dos ángulos, separados por una coma, se expresan en formato decimal, es decir, un número positivo o negativo con uno o más decimales. El primer ángulo representa la latitud y el segundo la longitud. Cuando el valor del ángulo es positivo se puede incluir el símbolo +. Cuando es negativo siempre se debe incluir el símbolo - delante. Puede haber espacios entre los valores y la coma, pero no entre los valores y los símbolos + y -.

Ejemplos:

```
30.0, -40.5
-25.05, +15.123
```

Sexagesimal

Los dos ángulos, separados por una coma, se expresan en formato sexagesimal incluyendo grados, minutos y segundos. El primer ángulo representa la latitud y el segundo la longitud. Tanto los grados como los minutos serán valores enteros mientras que los segundos serán un valor real con cuatro decimales. Se usan los símbolos ° (ASCII 167, UTF8 176), ‘ (ASCII y UTF8 39) y " (ASCII y UTF8 34) para acompañar a los valores que representan los grados, minutos y segundos respectivamente. Los ángulos serán siempre positivos y tras cada uno se incluye una letra para indicar la orientación. Para la latitud N o S y para la longitud E o W. Puede haber cero o más espacios entre los valores y la coma, entre los valores y las letras N, S, E o W y entre los propios valores. Nunca habrá espacio entre los números y los símbolos usados para grados, minutos y segundos.

Ejemplos:

```
30° 0‘ 0.0000" N, 40° 30‘ 0.0000" W
25° 3‘ 0.0000" S, 15° 7‘ 22.8000" E
```

GPS

Los dos ángulos se expresan en formato sexagesimal pero sin incluir los símbolos °, ‘ ni " ni usar espacios entre ellos. Se usan tres dígitos para indicar los grados, dos para los minutos (rellenando con ceros cuando sea necesario), y dos dígitos un punto y cuatro decimales para indicar los segundos. Los ángulos serán siempre positivos y tras cada uno se incluye una letra para indicar la orientación. Para la latitud N o S y para la longitud E o W.

Ejemplos:

```
0300000.0000N0403000.0000W
0250300.0000S0150722.8000E
```

Anexo 3. Sintaxis extendida de ER

La manera de escribir expresiones regulares varía de unas aplicaciones a otras, aunque hay un núcleo común en las distintas variantes de sintaxis de las expresiones regulares (*regex flavors*). En este anexo se introduce la sintaxis más común usada en lenguajes de programación como Python que es una extensión de la sintaxis teórica que se estudia en teoría. La sintaxis que se usa en algunos programas como *egrep, vi* o *flex* que forman parte de la distribución de Linux es algo diferente.

Cualquier aplicación que permite usar expresiones regulares para realizar operaciones de *regular pattern matching*, como validación de formato de cadenas, búsqueda o sustitución usando un patrón dado por una ER, cuenta con un *Motor de ER (regex engine). Este* se encarga de compilar o traducir la ER en una implementación de autómata finito cuya tabla de transición se usa en los métodos para procesar las cadenas. Dicha implementación es transparente al usuario/programador, que se ocupa sólo de escribir la ER y de aplicar los métodos que proporciona el *regex engine* para resolver los problemas de procesamiento de cadenas de patrón regular. Lo más importante es tener en cuenta que el proceso de compilación es mucho más costoso en tiempo que el uso de la ER ya compilada.

Funcionamiento básico de EditPad Pro

EditPad Pro es un editor de textos que cuenta con un regex engine. Puede descargarse de:

http://www.editpadpro.com/download.html (free trial)

En recursos del Aula Virtual hay ficheros que pueden usarse para probar los ejemplos de ER que estudiaremos a continuación. Por ejemplo:

- Abre el fichero *prueba-EditPad-varios.html,*
- Selecciona en el menú *Search* la opción *Show Search Panel.*
- Selecciona en el panel *Search&Replace* las casillas *Regular Expression y Case Sensitive.*
- Con *Find First* busca en el fichero cargado **la primera cadena del texto S coincidente con la expresión regular R** introducida en *Search* (primera cadena S tal que *R casa con S*) y la resalta en gris.
- Con *Next* busca la siguiente cadena coincidente **desde la posición en que finalizó el *matching* anterior**. Si en el texto no hay ninguna cadena que se ajuste al patrón de R, el editor lo advierte con una cruz roja.
- En la barra de *Search&Replace* se puede activar *Highlight*. Esto hace que se resalten **todas las cadenas coincidentes**, como si se hiciera *Next* de forma repetida. El resalte en algunos casos puede resultar confuso si se solapan las coincidencias.

Precaución con espacios en blanco: en el panel *Search&Replace* hay que tener cuidado de no poner espacios en blanco y otros caracteres que no sean parte del patrón de la ER, sobre todo al hacer *copy-paste* de Word a EditPad. Los blancos los señala con un punto gris claro para evitar confusión.

Tabla de sintaxis extendida

A continuación, introducimos un resumen de la sintaxis extendida de ER.

Expresiones regulares que describen un único carácter o un conjunto de caracteres		
Sintaxis	Descripción	Explicación y ejemplos de emparejamiento (Expresión regular en azul y cadena literal en amarillo)
c	c representa un carácter literal, que es cualquier carácter que no sea uno de los siguientes caracteres especiales para la sintaxis `[ ] \ ^ $ . | ? * + ( ) { }`	a casa con a Una ER que consiste en una secuencia de caracteres literales casa exactamente con la cadena que forma la ER: hola casa con hola (y ninguna más en ***modo case sensitive***, que diferencia mayúsculas de minúsculas) hola no casa con holaMundo ni con HoLA
\e	e representa un carácter especial (listados arriba) y al poner \e, la barra de escape hace que el carácter especial e pierda su significado y entonces \e describe al carácter literal e.	8\+2 casa con 8+2 (y ninguna más) 8+2 no casa con 8+2
\n \r \t	ERs para describir caracteres de LF (*line feed*), CR (*carrige return*) y tabuladores, respectivamente.	La ER \r\n describe a un *newline* en Windows y \n a un *newline* en Unix/Linux
[secCar]	[secCar] describe un conjunto de caracteres literales. La ER casa con cualquier carácter individual que aparezca en la secuencia de caracteres secCar. Los caracteres especiales pierden su significado dentro de los corchetes, pasan a ser literales y no hay que ponerles la barra de escape.	[hola] casa con h (también los caracteres o, l, y a y ninguno más) [hola] no casa con hola [hH][oO][Ll][aA] casa con hola y también con HoLa. Es decir sólo con la palabra "hola" con letras mayúsculas o minúsculas. [+*] casa únicamente con + o con *
[^secCar]	Al poner ^ (gorro o caret) se complementa el conjunto de caracteres que va detrás con respecto al conjunto total de caracteres.	[^+*] casa con cualquier carácter que NO sea + ni *
[**c1-c2**]	[c1-c2] describe un conjunto de caracteres dado por un rango y casa con cualquier carácter individual cuyo código ASCII esté comprendido entre el código del carácter c1 y el código del carácter c2. Pueden aparecer varios rangos dentro de los corchetes y se puede complementar el rango con ^	[0-9] equivale a la ER (0|1|2|3|4|5|6|7|8|9) que casa con cualquier dígito decimal [^0-9] casa con cualquier carácter que NO sea un dígito [a-z] casa con cualquier letra minúscula ASCII [a-zA-Z] es una ER con dos rangos y casa

		con una letra mayúscula o minúscula ASCII [a-z][a-z][0-9] casa pe2, zb7,… y no casa con e2 ni con variable1
\d	La ER \d es una abreviatura (*sorthand*) que describe al conjunto de los dígitos decimales.	\d equivale a [0-9] \d\d\d casa con 885 y no casa con 12
\w	La ER \w es una abreviatura que describe al conjunto de los *caracteres tipo word (palabra):* son los dígitos decimales, letras ASCII y _ (subrayado).	\w equivale a [0-9a-zA-Z_] \w[\d.,] casa con h1 con r, y con 9. (las abreviaturas pueden usarse dentro de corchetes)
\p{L} **\p{Lu}** **\p{Ll}**	\p{L} describe al conjunto de letras Unicode (mayúsculas o minúsculas). Permite que las letras acentos, diéresis, signos fonéticos, etc. \p{Lu} sólo para letras Unicode mayúsculas \p{Ll} sólo para letras Unicode minúsculas	La ER [a-zA-Z]+ no casa con ambigüedad ni con España ni con fácil. \p{L}+ casa con cualquiera de las anteriores. \p{Ll}+ casa con fácil y no casa con España \p{Lu}+ casa con ESPAÑA y no casa con España
\s	La ER \s es una abreviatura que describe el conjunto de *caracteres de espaciado* (blanco, \t, \r o \n). Equivale a [\n\r\t]	hola[\d\s]mundo casa con hola1mundo con hola mundo y no casa con holamundo
.	El punto (dot) es una ER que casa con cualquier carácter *excepto* los caracteres de nueva línea \r o \n	car. casa con cara caro car1 car. car+ car\. casa con car. (y ninguna cadena más)
Expresiones regulares con operador de alternancia y paréntesis de agrupación (*alternation, grouping*)		
R1|R2	El | (pipe) es el operador de unión o alternancia de ER. La expresión **R1**|R2 tiene coincidencia con cadenas que se ajustan a R1 o a R2. Es el operador de menor precedencia y con los paréntesis se agrupa una subexpresión para alterar la precedencia.	America|Europe|Canada casa con America con Europe con Canada y con ninguna cadena más. (hola|bye)Mundo casa con holaMundo con byeMundo y con ninguna más. (hola|bye)Mundo equivale, por la propiedad distributiva, a holaMundo|byeMundo pero es mejor la primera forma.
Expresiones regulares con operadores de opcionalidad y repetición (*quantifiers*)		

R*	La estrella * es el operador de clausura. R* indica cero o más repeticiones de cadenas que casan con el patrón R	[a-z]\d* casa con a1234 con b2 con c y no casa con 12 ni con d-83
R*?	*? es el operador de clausura perezosa (*lazy star*). Cuando se usa R*? en una búsqueda, el *regex engine* selecciona la cadena más corta que se ajusta al patrón R*.	Buscando con <.*> en una línea que contiene <h3>CHARACTER SET</h3> y sin más ángulos > en línea, se selecciona <h3>CHARACTER SET</h3> Buscando con <.*?> selecciona desde <h3>. Es decir, sólo hasta el primer ángulo de cierre.
R+	Se corresponde con el operador de clausura positiva de lenguajes. R+ indica una o más repeticiones del patrón R. Equivale a RR*	[a-z]\d+ casa con a1234 con b2 y no casa con c ni con 12 ni con d-83
R+?	+? es versión perezosa del operador + (*lazy plus*). Análogo a R*? pero obligando al menos una coincidencia con R	Buscando con <.+?> en una línea que contiene <> </br> y sin más > en la línea, se selecciona <> </br> Si se busca con <.*?> entonces se selecciona <>
R?	El operador ? (no precedido de * ni +) es un operador de opcionalidad e indica que R es opcional. R? equivale a (R|λ)	abc? casa con ab y con abc y con ninguna cadena más a(bc)? casa con a y con abc y con ninguna cadena más
R{n} R{n,m} R{n,}	Indica n repeticiones del patrón R Indica entre n y m repeticiones de R Indica n o más repeticiones de R	a{3} casa con aaa y con ninguna cadena más v\d{1,3} casa con v5 y con v60 y con v623 y no casa con v ni con v8965 [A-Z]{3}\d{2,} casa con BXZ47 y con ABC123 y con RSA1234 y no casa con B43 ni con 576
ER con anclas y delimitadores de palabras *(anchors, word boundaries)*		
^R	EL ^ cuando va fuera de corchetes tiene coincidencia con la posición de comienzo de una cadena o línea de texto. Cuando se usa ^R hay matching con una cadena S que se ajusta al patrón R y aparece como prefijo.	Buscando con ^<[a-zA-Z]+> se selecciona en una línea que comience por Si antes del carácter '<' hay espacios u otro carácter entonces no hay *matching* en esa línea.
R$	Análogo al anterior pero con la posición de final de cadena o línea de texto.	Buscando con <[a-zA-Z]+>$ se selecciona <head> en una línea que acaba por <head> Buscando con la ER ^\d+$ selecciona cadenas de uno o más dígitos que ocupan toda la línea.
\bR\b	\b es un delimitador de palabra y en una búsqueda con \bR\b se selecciona una	Buscando con \b\d+\b se selecciona 8859 en una línea que contiene charset=ISO-

	cadena que casa con R y está delimitada a la izquierda y a la derecha por caracteres que NO son letras o dígitos.	`8859-` Si fuera `charset=ISO8859-` no habría matching con `8859` Se usa el ancla `\b` para emparejamiento con palabras completas.
ER con operadores de contexto (*lookahead y lookbehind*)		
`R1(?=R2)`	*Positive lookahead.* Busca cadenas que verifican R1 siempre y cuando vayan seguidas de cadenas que verifican R2. La cadena coincidente con la expresión regular completa es sólo la que verifica R1.	`[a-z]+(?=@)` `selecciona alf@um.es`
`R1(?!R2)`	*Negative Lookahead*. Busca cadenas que verifican R1 siempre y cuando no vayan seguidas de cadenas que verifican R2. La cadena coincidente con la expresión regular completa es sólo la que verifica R1.	`[a-z]+(?![a-z@])` `selecciona alf@um.es`
`(?<=R2)R1`	*Positive Lookbehind*. Busca cadenas que verifican R1 siempre y cuando vayan precedidas de cadenas que verifican R2. La cadena coincidente con la expresión regular completa es sólo la que verifica R1.	`(?<=\.)[a-z]+` `selecciona alf@um.es`
`(?<!R2)R1`	*Negative Lookbehind*. Busca cadenas que verifican R1 siempre y cuando no vayan precedidas de cadenas que verifican R2. La cadena coincidente con la expresión regular completa es sólo la que verifica R1.	`(?<![.a-z])[a-z]+` `selecciona alf@um.es`

Modificadores de expresiones regulares

Los modificadores de las expresiones regulares sirven para alterar el modo en que el regex engine de cierta herramienta realiza las operaciones de *pattern matching* a partir de una expresión regular y de ese modo se puede acortar la expresión regular. Sólo deben usarse cuando realmente tengan algún efecto, no de forma indiscriminada, pues disminuyen la eficiencia.

Modificador para modo "dot matches newline"

En métodos con ER en Python y en editores de texto como EditPad Pro (ver sección de funcionamiento de EditPad más adelante), el punto (*dot*) no tiene coincidencia por defecto con los caracteres de nueva línea (como \r o \n).

Ejemplo: si se usa la ER </head>.*<BODY> para buscar en un texto donde aparecen las etiquetas html </head> y luego <BODY> separadas por newlines entonces el *regex engine* no encuentra ninguna cadena coincidente.

Para que el punto tenga coincidencia con caracteres de nueva línea (modo *dot matches newline)*, sin que dependa de la herramienta concreta, se usa el *modificador de dot* **(?s)** delante de laER.

Ejemplo: con (?s)</head>.*<BODY> se selecciona desde </head> hasta <BODY> incluido, aunque esas etiquetas html vayan en líneas separadas.

Modificador para modo "case insensitive"

El modo por defecto en los métodos con ER en Python es *case sensitive*, esto es, se distingue entre mayúsculas y minúsculas en las letras en las operaciones de matching. Para activar el modo contrario *case insensitive* para una ER sin que dependa de la herramienta concreta se pone el *modificador de case* **(?i)** delante de laER.

Ejemplo: `(?i)hola casa con HoLA hola HOLA (no se distinguen mayúsculas de minúsculas)`

Mezcla de modificadores

Los modificadores pueden juntarse, se puede poner **(?is)R** para indicar modo *insensitive* y *dot matches newlines* para la expresión regular R que va detrás de los modificadores.

Ejemplo: Con (?is)</head>.*<body> se selecciona desde la etiqueta </head> hasta <body> incluido, aunque esas etiquetas vayan en líneas separadas y con independencia de que las palabras de etiqueta "head" y "body" vayan con las letras en mayúsculas o minúsculas. Tiene sentido poner el modificador de case porque en el lenguaje Html no se diferencian las mayúsculas de las minúsculas en las etiquetas. La búsqueda también tendría éxito si, por ejemplo, la palabra clave "head" aparece en el documento escrita como "Head" o "HEAD".

Capturas y referencias a capturas (backreferences)

Cada una de las subexpresiones **entre paréntesis** que aparecen dentro de una expresión regular forma un **grupo**. Los grupos se enumeran desde 1 contando los paréntesis de apertura en una ER de izquierda a derecha y el grupo 0 siempre es la propia expresión regular, aunque no vaya entre paréntesis.

Ejemplo: En la ER ([A-Z]+)(\d+)([a-z]+) aparecen paréntesis que no son necesarios para alterar la precedencia, pero al ponerlos se distinguen 4 grupos en la expresión regular:

El **grupo 0** siempre es la propia expresión regular, en este caso ([A-Z]+)(\d+)([a-z]+)

El **grupo 1** consiste en la subexpresión regular: ([A-Z]+)

El **grupo 2** es: (\d+)

El **grupo 3** es: ([a-z]+)

Los grupos de una ER pueden ser usados para que el *regex engine* haga una captura de las subcadenas que se ajustan al patrón de cada grupo, proceso al que nos referimos de forma abreviada como **captura de grupos**. Las capturas **pueden referenciarse** mediante la referencia

\i donde i es el número del grupo, por ej. \1 (captura de grupo 1), \3 (captura de grupo 3), o bien \0 para la captura de la cadena que se ajusta a la ER completa.

Ejemplo: Usando la ER ([A-Z]+)(\d+)([a-z]+) con la cadena AULA12b:

El **grupo 0** AULA12b

El **grupo 1** AULA

El **grupo 2** es: 12

El **grupo 3** es: b

¿Cómo se usan las referencias a las capturas para hacer sustituciones con ER?

Una de las aplicaciones del proceso de captura de grupos de una ER es para realizar una sustitución que no es por cadena fija (la típica), sino que se sustituye por una cadena que depende del patrón de la ER.

Ejemplo: Supongamos que queremos buscar una cadena que se ajusta al patrón ([A-Z]+)(\d+)([a-z]+) para sustituirla por la cadena que consiste en las letras minúsculas de la cadena original seguidas de un guion, luego los dígitos encerrados entre corchetes, seguido de un guion y finalmente las letras mayúsculas. Por ejemplo para sustituir AB12xy por xy-[12]-AB.

Obviamente esto no se puede hacer con un *replace* o sustitución simple como se hace con los procesadores de texto de uso general. En el editor EditPad, que es un editor profesional para programadores y permite usar ER, se pondría lo siguiente en el panel *search&replace*:

SEARCH: ([A-Z]+)(\d+)([a-z]+) REPLACE: \3-[\2]-\1

Con *next+replace* sucesivas veces podemos sustituir cada cadena coincidente con la ER del campo *Search*. Con *Replace All* se sustituyen de golpe todas las cadenas coincidentes.

Es importante tener en cuenta que la cadena sustituida es literal excepto por las referencias a los grupos capturados.

También es posible nombrar los grupos y utilizar los nombres en lugar de sus posiciones a la hora de indicar los reemplazos. Para nombrarlo se incluye ?P seguido del identificador entre los símbolos < y > al principio del grupo. Después, podemos referenciar los grupos capturados con \g<id> siendo id el identificador del grupo.

Ejemplo: Si vamos buscando la planta y la letra de una vivienda en una dirección con la ER (\d+)([a..zA..Z]) podríamos darle nombre a cada uno de los dos grupos así: (?P<planta>\d+)(?P<letra>[a..zA..Z]). De este modo, la sustitución podría hacerse así:

SEARCH: (?P<planta>\d+)(?P<letra>[a..zA..Z]) REPLACE: Piso \g<planta> letra \g<letra>

Finalmente, cuando se quieren usar paréntesis para agrupar partes de ER pero no se quiere que dichos grupos puedan ser referenciados, basta con añadir la secuencia :? a continuación del primer paréntesis.

Ejemplo: Buscando con la ER ([a..z]+)(:?\d+)([a..z]+) los grupos posibles a usar, además del 0, serían el 1 y el 2 que coincidirían con las cadenas de 1 o más letras minúsculas. Pero los dígitos que las separan no podrían referenciarse.

Uso de referencias dentro de una ER para búsquedas más complejas

Las referencias permiten construir **expresiones regulares aumentadas más potentes** (describen lenguajes que no son regulares). Este tipo de expresiones regulares aumentadas no es traducible a un autómata finito puro, porque se requiere guardar en memoria la captura (parte de la cadena que se procesa) y el modelo de autómata finito es el de una máquina sin memoria.

Ejemplo: Supongamos que queremos buscar en un texto dos palabras repetidas, entendiendo que son cadenas iguales formadas por letras Unicode mayúsculas o minúsculas y las palabras van separadas por uno o más caracteres de tipo espacio (\s).

Por ejemplo, "HOLA Pedro Hola Juan" debería seleccionase todo el texto entre los dos saludos. Y da lo mismo que vayan en la misma línea o en líneas separadas y que una de las palabras lleve minúsculas. También requerimos que se seleccionen **palabras completas**, no fragmentos de palabras. Por ejemplo, si en una línea aparece "es un uno" entonces no se debería resaltar "Es un uno" porque "un" forma parte de una palabra, no es una palabra completa. Esto indica que debemos usar el delimitador de palabra \b. En EditPad pondríamos la siguiente ER para la búsqueda

SEARCH: \b(\p{L}+)\b\s+\b\1\b