PROGRAMACIÓN II - UNIDAD 4

Ing. Gastón Weingand (gaston.weingand@uai.edu.ar)

Agenda

1. Entrada y salida estándar.
2. Archivos: Lectura y escritura.
3. Cierre y tratamiento de errores en la gestión de archivos.
4. Expresiones regulares, patrones. Módulo re.

Agenda

1. Entrada y salida estándar.
2. Archivos: Lectura y escritura.
3. Cierre y tratamiento de errores en la gestión de archivos.
4. Expresiones regulares, patrones. Módulo re.

4- Expresiones regulares

Las expresiones regulares, también llamadas regex o regexp, consisten en patrones que describen conjuntos de cadenas de caracteres. Algo parecido sería escribir en la línea de comandos de Windows dir *.exe

'*.exe' sería una "expresión regular" que describiría todas las cadenas de caracteres que empiezan con cualquier cosa seguida de '.exe', es decir, todos los archivos exe.

4- Expresiones regulares

Módulo re

El módulo re contiene funciones para buscar patrones dentro de una cadena (search), comprobar si una cadena se ajusta a un determinado criterio descrito mediante un patrón (match), dividir la cadena usando las ocurrencias del patrón como puntos de ruptura (split) o para sustituir todas las ocurrencias del patrón por otra cadena (sub). Veremos estas funciones y alguna más en la próxima sección, pero por ahora, aprendamos algo más sobre la sintaxis de las expresiones regulares.

4- Expresiones regulares

Patrones

La expresión regular más sencilla consiste en una cadena simple, que describe un conjunto compuesto tan solo por esa misma cadena. Por ejemplo, veamos cómo la cadena “python” coincide con la expresión regular “python” usando la función match:

```
import re
if re.match("python", "python"):
    print ("cierto")
```

4- Expresiones regulares

Patrones

Si quisiéramos comprobar si la cadena es “python”, “¡ython”, “cython” o cualquier otra cosa que termine en “ython”, podríamos utilizar el carácter comodín, el punto ‘.’:

```
if re.match(".ython", "python"):
    print("cierto")

if re.match(".ython", "jython"):
    print("cierto")
```

4- Expresiones regulares

Patrones

Si necesitáramos una expresión que sólo resultara cierta para las cadenas “python”, “jython” y “cython” y ninguna otra, podríamos utilizar el carácter ‘|’ para expresar alternativa escribiendo los tres subpatrones completos:

```
if re.match("python|jython|cython", "python"):
    print("cierto")
```

4- Expresiones regulares

Patrones

o bien tan solo la parte que pueda cambiar, encerrada entre paréntesis, formando lo que se conoce como un grupo. Los grupos tienen una gran importancia a la hora de trabajar con expresiones regulares y este no es su único uso, como veremos en la siguiente sección.

```
if re.match("(p|j|c)ython", "python"):
    print("cierto")
```

4- Expresiones regulares

Patrones

¿Y si quisiéramos comprobar si la cadena es “python0”, “python1”, “python2”, ... , “python9”? En lugar de tener que encerrar los 10 dígitos dentro de los corchetes podemos utilizar el guión, que sirve para indicar rangos. Por ejemplo, a-d indicaría todas las letras minúsculas de la ‘a’ a la ‘d’; 0-9 serían todos los números de 0 a 9 inclusive.

```
if re.match("python[0-9]", "python0"):
    print("cierto")
```

4- Expresiones regulares

Patrones

Si quisiéramos, por ejemplo, que el último carácter fuera o un dígito o una letra simplemente se escribirían dentro de los corchetes todos los criterios, uno detrás de otro.

```
if re.match("python[0-9a-zA-Z]", "pythonp"):
    print("cierto")
```

4- Expresiones regulares

Patrones

Es necesario advertir que dentro de las clases de caracteres los caracteres especiales no necesitan ser escapados. Para comprobar si la cadena es “python.” o “python,”, entonces, escribiríamos:

```
if re.match("python[.,]", "python."):
    print("cierto")
```

y no

```
if re.match("python[\.,]", "python."):
    print("cierto")
```

ya que en este último caso estaríamos comprobando si la cadena es “python.”, “python,” o “python\”.

4- Expresiones regulares

Patrones

Los conjuntos de caracteres también se pueden negar utilizando el símbolo '^'. La expresión "python[^0-9a-z]", por ejemplo, indicaría que nos interesan las cadenas que comiencen por "python" y tengan como último carácter algo que no sea ni una letra minúscula ni un número.

```
if re.match("python[^0-9a-z]", "python+"):
    print("cierto")
```

4- Expresiones regulares

Patrones

El uso de [0-9] para referirse a un dígito no es muy común, ya que, al ser la comprobación de que un carácter es un dígito algo muy utilizado, existe una secuencia especial equivalente: '\d'. Existen otras secuencias disponibles que listamos a continuación:

- "\d": un dígito. Equivale a [0-9]
- "\D": cualquier carácter que no sea un dígito. Equivale a [^0-9]
- "\w": cualquier caracter alfanumérico. Equivale a [a-zA-Z0-9_]
- "\W": cualquier carácter no alfanumérico. Equivale a [^a-zAZ0-9_]
- "\s": cualquier carácter en blanco. Equivale a [\t\n\r\f\v]
- "\S": cualquier carácter que no sea un espacio en blanco. Equivale a [^ \t\n\r\f\v]

4- Expresiones regulares

Patrones

Veamos ahora cómo representar repeticiones de caracteres, dado que no sería de mucha utilidad tener que, por ejemplo, escribir una expresión regular con 30 caracteres '\d' para buscar números de 30 dígitos. Para esta situación tenemos los caracteres especiales +, * y ?, además de las llaves {}.

El carácter + indica que lo que tenemos a la izquierda, sea un carácter como 'a', una clase como '[abc]' o un subpatrón como (abc), puede encontrarse una o mas veces. Por ejemplo, la expresión regular "python+" describiría las cadenas "python", "pythonn" y "pythonnn", pero no "pytho", ya que debe haber al menos una n.

4- Expresiones regulares

Patrones

El carácter * es similar a +, pero en este caso lo que se sitúa a su izquierda puede encontrarse cero o mas veces.
El carácter ? indica opcionalidad, es decir, lo que tenemos a la izquierda puede o no aparecer (Puede aparecer 0 o 1 veces).

Finalmente, las llaves sirven para indicar el número de veces exacto que puede aparecer el carácter de la izquierda, o bien un rango de veces que puede aparecer. Por ejemplo {3} indicaría que tiene que aparecer exactamente 3 veces, {3,8} indicaría que tiene que aparecer de 3 a 8 veces, {,8} de 0 a 8 veces y {3,} tres veces o mas (las que sean).

4- Expresiones regulares

Patrones

Otro elemento interesante en las expresiones regulares, para terminar, es la especificación de las posiciones en que se tiene que encontrar la cadena, esa es la utilidad de ^ y $, que indican, respectivamente, que el elemento sobre el que actúan debe ir al principio de la cadena o al final de esta. La cadena “http://mundogeek.net”, por ejemplo, se ajustaría a la expresión regular “^http”, mientras que la cadena “El protocolo es http” no lo haría, ya que el http no se encuentra al principio de la cadena.

4- Expresiones regulares

Modulo re

Ya hemos visto por encima cómo se utiliza la función match del módulo re para comprobar si una cadena se ajusta a un determinado patrón. El primer parámetro de la función es la expresión regular, el segundo, la cadena a comprobar y existe un tercer parámetro opcional que contiene distintos flags que se pueden utilizar para modificar el comportamiento de las expresiones regulares.

4- Expresiones regulares

Modulo re

Algunos ejemplos de flags del módulo re son re.IGNORECASE, que hace que no se tenga en cuenta si las letras son mayúsculas o minúsculas o re.VERBOSE, que hace que se ignoren los espacios y los comentarios en la cadena que representa la expresión regular.

4- Expresiones regulares

Modulo re

El valor de retorno de la función será None en caso de que la cadena no se ajuste al patrón o un objeto de tipo MatchObject en caso contrario. Este objeto MatchObject cuenta con métodos start y end que devuelven la posición en la que comienza y finaliza la subcadena reconocida y métodos group y groups que permiten acceder a los grupos que propiciaron el reconocimiento de la cadena.

4- Expresiones regulares

Modulo re

Al llamar al método group sin parámetros se nos devuelve el grupo 0 de la cadena reconocida. El grupo 0 es la subcadena reconocida por la expresión regular al completo, aunque no existan paréntesis que delimiten el grupo.

```
mo = re.match("http://.+net", "http://mundogeek.net")
print(mo.group())
```

4- Expresiones regulares

Modulo re

Podríamos crear grupos utilizando los paréntesis, como aprendimos en la sección anterior, obteniendo así la parte de la cadena que nos interese.

```
mo = re.match("http://(.+).net", "http://mundogeek.net")
print(mo.group(0))
print(mo.group(1))
```

4- Expresiones regulares

Modulo re

El método groups, por su parte, devuelve una lista con todos los grupos, exceptuando el grupo 0, que se omite.

```
mo = re.match("http://(.+)(.{3})", "http://mundogeek.net")
print (mo.groups())
```

4- Expresiones regulares

Modulo re

- La función search del módulo re funciona de forma similar a match; contamos con los mismos parámetros y el mismo valor de retorno. La única diferencia es que al utilizar match la cadena debe ajustarse al patrón desde el primer carácter de la cadena, mientras que con search buscamos cualquier parte de la cadena que se ajuste al patrón. Por esta razón el método start de un objeto MatchObject obtenido mediante la función match siempre devolverá 0, mientras que en el caso de search esto no tiene por qué ser así.
- Otra función de búsqueda del módulo re es findall. Este toma los mismos parámetros que las dos funciones anteriores, pero devuelve una lista con las subcadenas que cumplieron el patrón.

4- Expresiones regulares

Modulo re

Otra posibilidad, si no queremos todas las coincidencias, es utilizar finditer, que devuelve un iterador con el que consultar uno a uno los distintos MatchObject. Las expresiones regulares no solo permiten realizar búsquedas o comprobaciones, sino que, como comentamos anteriormente, también tenemos funciones disponibles para dividir la cadena o realizar reemplazos.

4- Expresiones regulares

Modulo re

- La función split sin ir más lejos toma como parámetros un patrón, una cadena y un entero opcional indicando el número máximo de elementos en los que queremos dividir la cadena, y utiliza el patrón a modo de puntos de separación para la cadena, devolviendo una lista con las subcadenas.
- La función sub toma como parámetros un patrón a sustituir, una cadena que usar como reemplazo cada vez que encontremos el patrón, la cadena sobre la que realizar las sustituciones, y un entero opcional indicando el número máximo de sustituciones que queremos realizar.

4- Expresiones regulares

Modulo re

Al llamar a estos métodos lo que ocurre en realidad es que se crea un nuevo objeto de tipo RegexObject que representa la expresión regular, y se llama a métodos de este objeto que tienen los mismos nombres que las funciones del módulo.

Para crear un objeto RegexObject se utiliza la función compile del módulo, al que se le pasa como parámetro la cadena que representa el patrón que queremos utilizar para nuestra expresión regular y, opcionalmente, una serie de flags de entre los que comentamos anteriormente.