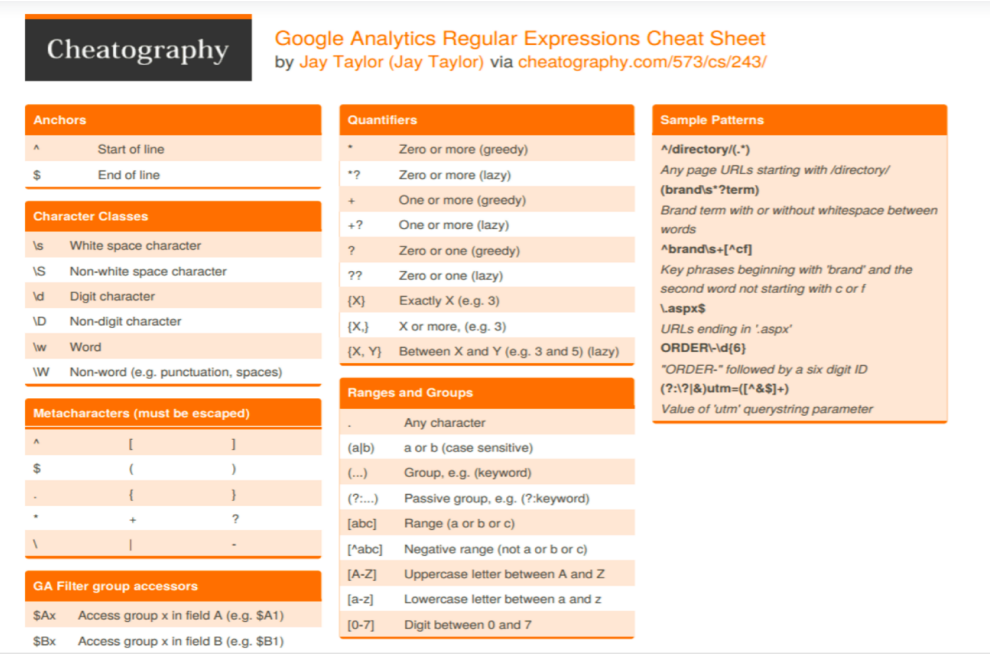
UT 6 y 7: Expresiones regulares

Una expresión regular, o *regex*, define un patrón de búsqueda para cadenas de caracteres. La podemos utilizar para comprobar si una cadena contiene o coincide con el patrón.

Este tipo de mecanismos se utilizan en todos los lenguajes y siguen unas normas y "operadores" comunes:



Cheatography — Google Analytics Regular Expressions Cheat Sheet
by Jay Taylor (Jay Taylor) via cheatography.com/573/cs/243/

Anchors

^	Start of line
$	End of line

Character Classes

\s	White space character
\S	Non-white space character
\d	Digit character
\D	Non-digit character
\w	Word
\W	Non-word (e.g. punctuation, spaces)

Metacharacters (must be escaped)

^	[	]
$	(	)
.	{	}
*	+	?
\	|	-

GA Filter group accessors

$Ax	Access group x in field A (e.g. $A1)
$Bx	Access group x in field B (e.g. $B1)

Quantifiers

*	Zero or more (greedy)
*?	Zero or more (lazy)
+	One or more (greedy)
+?	One or more (lazy)
?	Zero or one (greedy)
??	Zero or one (lazy)
{X}	Exactly X (e.g. 3)
{X,}	X or more, (e.g. 3)
{X, Y}	Between X and Y (e.g. 3 and 5) (lazy)

Ranges and Groups

.	Any character
(a|b)	a or b (case sensitive)
(...)	Group, e.g. (keyword)
(?:...)	Passive group, e.g. (?:keyword)
[abc]	Range (a or b or c)
[^abc]	Negative range (not a or b or c)
[A-Z]	Uppercase letter between A and Z
[a-z]	Lowercase letter between a and z
[0-7]	Digit between 0 and 7

Sample Patterns

^/directory/(.*)
Any page URLs starting with /directory/
(brand\s*?term)
Brand term with or without whitespace between words
^brand\s+[^cf]
Key phrases beginning with 'brand' and the second word not starting with c or f
\.aspx$
URLs ending in '.aspx'
ORDER\-\d{6}
"ORDER-" followed by a six digit ID
(?:\?|&)utm=([^&$]+)
Value of 'utm' querystring parameter

Símbolos	*Significado*
`.`	Cualquier carácter.
`\d`	Cualquier dígito del **0** al **9**.
`\D`	Cualquier carácter que no sea un dígito del **0** al **9**.
`x*`	**x** puede aparecer cero o más veces.
`x+`	**x** debe aparecer al menos una vez.
`x?`	**x** puede aparecer una vez o no aparecer.
`[abc]` o `[a|b|c]`	Cualquier carácter indicado entre los corchetes: **a**, **b** o **c**.
`[a-z]`	Cualquier carácter de la **a** a la **z**.
`x{n}`	**x** debe aparecer **n** veces.
`x{n,m}`	**x** debe aparecer entre **n** y **m** veces.
`x{n,}`	**x** debe aparecer al menos **n** veces.

Para poder utilizar *regex* en Java se necesita utilizar algunas clases de su API: *Pattern* y *Matcher.*

- La clase ***Pattern*** nos permite definir un patrón de búsqueda y generar un objeto de *Matcher*.

- ***Matcher*** nos permite realizar las búsquedas oportunas.

```java
public static void main(String[] args) {

    String input = "Hola que tal estas hola, hola2";

    Pattern patron = Pattern.compile("[Hh]ola");
    Matcher mat = patron.matcher(input);

    // El método find nos indica si se han encontrado o no subsecuencias del patrón definido
    while (mat.find()) {
        System.out.println(mat.group() + " - posición inicial: " + mat.start() + ", posición final:" + mat.end());
    }

    // El método matches busca una concordancia completa entre patrón y búsqueda
    // Hay dos formas de ejecutarlo:

    System.out.println(Pattern.matches("[Hh]ola", input));
    System.out.println(mat.matches());

    // En ambos casos daría falso puesto que con ese regex solo encuentra
    // subsecuencias y no hay una coincidencia completa.

    // Para que diera true, tendríamos que indicar que empieza por H o h, seguido de
    // "ola", y después, vendrían cualquier caracter un número indefinido de veces:
    System.out.println(Pattern.matches("^[Hh]ola.*", input));

    // Otro uso sería buscar un patrón como divisor de cadenas.
    // En este caso el divisor se trata de cualquier dígito numérico que al menos
    // aparece 1 vez
    Pattern p = Pattern.compile("\\d+");
    String[] str = p.split(
            "Soy Fran. Mi número de empleado es: 456456, mi teléfono es: "
            + "0532214, mi correo electrónico es: aaa@aaa.com");
    for (String elem : str) {
        System.out.println(elem);
    }

}
```

```
Hola - posición inicial: 0, posición final:4
hola - posición inicial: 19, posición final:23
hola - posición inicial: 25, posición final:29
false
false
true
Soy Fran. Mi número de empleado es:
, mi teléfono es:
, mi correo electrónico es: aaa@aaa.com
```

Importante: Como algunos “operadores” empiezan por “\”, en Java no serían válidos porque podrían confundirse con otros caracteres especiales, como por ejemplo “\n”. Para poder utilizarlos, habría que “escapar” la barra atrás, duplicándola: “\\”, de esta forma se indica que la primera barra va a escapar otro carácter, que sería otra barra atrás.