Expresiones Regulares: Ejercicio I

1.1 - ¿Qué son las expresiones regulares?

Son una serie de concatenaciones de texto parecidas a las que se usan en programación, pero no son un lenguaje de programación, con el objetivo de buscar o manipular partes concretas en un texto.

Como por ejemplo *"\d"* que nos marcaría todos los dígitos que hay en un texto.

1.2 - Explica **brevemente** para qué sirven las expresiones regulares

---

Para buscar o manipular partes concretas dentro de un archivo de texto masivo de forma mucho más rápida y precisa.

1.3 - Ejercicio 3:

---

a. Desde la interfaz de comandos de UNIX, realiza los siguientes 3 ficheros:

- **file01.txt** → Esta es una prueba 1
- **file02.txt** → ESTA ES UNA PRUEBA 2
- **file03.txt** → esta es una prueba 3

```
localhost:~# ls
bench.py    hello.c     hello.js    readme.txt
localhost:~# cat > file01.txt
Esta es una prueba 1
^C
localhost:~# cat > file02.txt
ESTA ES UNA PRUEBA 2
^C
localhost:~# cat > file03.txt
esta es una prueba 3
^C
localhost:~# ls
bench.py    file02.txt  hello.c     readme.txt
file01.txt  file03.txt  hello.js
localhost:~#
```

b. Haz una búsqueda con grep para visualizar los ficheros que contengan el siguiente texto **"PRUEBA"** en la que deberá aparecer solamente el fichero **file02.txt**

Para la búsqueda en todos los ficheros usamos el * (global)

```
localhost:~# grep 'PRUEBA' *
file02.txt:ESTA ES UNA PRUEBA 2
localhost:~#
```

c. Haz una búsqueda con grep del siguiente texto **“prueba 3”** en la que deberá aparecer solamente el fichero **file03.txt**

```
localhost:~# grep 'prueba 3' *
file03.txt:esta es una prueba 3
localhost:~#
```

d. Haz una búsqueda con grep del siguiente texto **“prueba”** que sea **case-insensitive** en la que deberán aparecer los ficheros **file01.txt**, **file02.txt** y **file03.txt**

Con “-i” hacemos que no diferencie entre mayúsculas y minúsculas

```
localhost:~# grep -i 'prueba' *
file01.txt:Esta es una prueba 1
file02.txt:ESTA ES UNA PRUEBA 2
file03.txt:esta es una prueba 3
localhost:~#
```

e. Haz una búsqueda con grep que muestre solamente los ficheros que empiecen por **“ESTA”**. La búsqueda deberá devolver solamente el fichero **file02.txt**

“^ ” para indicar que queremos que nos imprima lo que empiece por ese valor.

```
localhost:~# grep '^ESTA' *
file02.txt:ESTA ES UNA PRUEBA 2
localhost:~#
```

f. Haz una búsqueda con grep que muestre solamente los ficheros que acaban con un **“3”**. La búsqueda deberá devolver solamente el fichero **file03.txt**

“ $” al final para que nos imprima lo que termina por ese valor.

```
localhost:~# grep '3$' *
file03.txt:esta es una prueba 3
localhost:~#
```

1.4 - ¿Qué es un motor de expresiones regulares? ¿Para qué sirven?

Son programas o bibliotecas que te permiten procesar e implementar patrones. Sirven para buscar, comparar o manipular patrones en los textos definidos mediante las expresiones regulares.

1.5 - ¿Cuáles son los principales motores de expresiones regulares? ¿Cuál vamos a utilizar nosotr@s?

Los principales motores son:

Oniguruma: Que está en Visual Studio Code, TextMate, Ruby, PHP entre otros.

Java (java.util.regexp): Un paquete de librerías para trabajar con las regexp. Las más utilizadas son Pattern y Matcher, que sirven para compilar regexp, buscar y realizar operaciones de reemplazo en cadena.

JavaScript (Regexp): Los navegadores y Node.js utilizan el regexp incorporado de JS a través de la clase RegExp. En JS las regexp se utilizan de forma nativa para hacer búsquedas o reemplazos en cadenas de texto.

GNU Grep (grep): grep es una comando para poder realizar búsquedas regexp en sistemas Unix.

PCRE (Perl Compatible Regular Expressions): Es un motor compatible con Perl, también se utiliza con re en PHP, Python y otros.

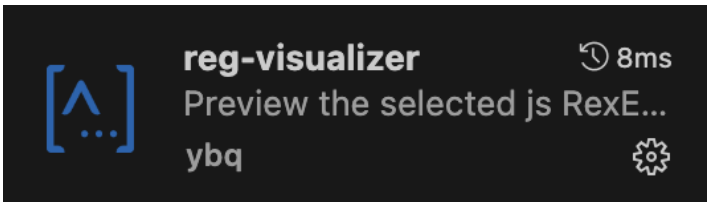
PHP (PCRE): PHP utiliza PCRE para las regexp. Por ejemplo se puede utilizar con la función preg_match() entre otras.

Ruby (Regexp): Ruby tiene soporte nativo mediante la clase Regexp.

C# (.NET Regex): C# Utiliza la clase System.Text.RegularExpressions.Regex para las regexp.

Python (re): Python lo incluye mediante el módulo re para buscar coincidencias o realizar cambios de forma parecida a Java.

Nosotr@s utilizaremos Visual Studio Code + el módulo *reg-visualizer* para trabajar con las expresiones regulares.

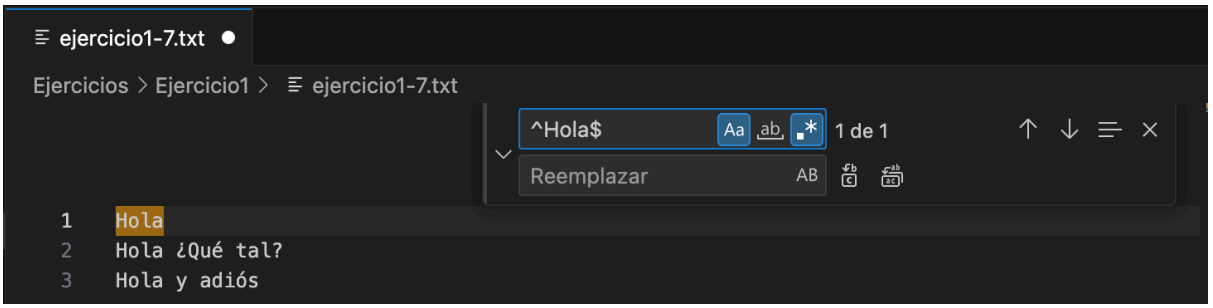


1.6 - ¿Qué es un patrón? ¿Y un match?

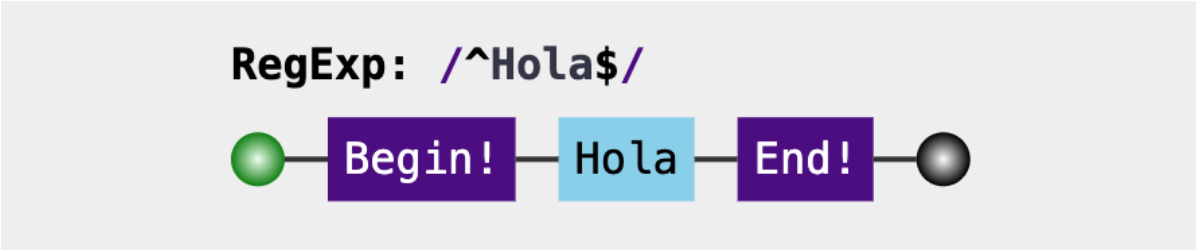
El patrón es una secuencia de caracteres mediante la cual intentaremos hacer un match dentro de un fichero.
Dicho de otra forma, buscar coincidencias de caracteres con las que hay dentro de un fichero utilizando un motor de expresiones regulares.

Un match es una coincidencia con lo que buscamos mediante un regexp y el contenido de un fichero.

1.7 - Saca el esquema del siguiente patrón ^Hola$ y, además, explica que hace dicha expresión regular sobre el siguiente documento:



Al sacar el esquema vemos lo siguiente:



con el carácter "^" al principio establecemos que el match tiene que empezar por *Hola*.
con el carácter "$" al final le decimos que el match tiene que terminar con *Hola*. Sin tener nada detrás de este.

Al juntar estos dos, le decimos que el match tiene que empezar con *Hola*, sin tener nada delante en esa línea, y a la vez ser *Hola* lo último de esa línea, osea no tener nada detrás.

Por eso solamente nos marca que hace match con la primera línea ya que las otras no cumplen la condición ("$") pues tienen texto detrás del Hola.