Las expresiones regulares son objetos que definen patrones que pueden cumplirse, o no, sobre un objeto de tipo String.

Es posible su declaración mediante notación literal, o mediante constructor:

```
/patrón/flags                       // literal


new RegExp(patrón [, flags])        // constructor
```

Sus usos más frecuentes son a través de los métodos `.match()` o `.test()` para verificar si el patrón se cumple en un String específico, y .replace() para sustituir los caracteres que coincidan con el patrón:

```
const text = "Ironhack"
text.match(/Ironhack/)              // true
```

#INTRODUCCIÓN

MÁS INFORMACIÓN

Puedes utilizarlas en la validación de tus formularios:

```
if (password.length < 7 || !password.match(/[A-Z]/) || !password.match(/[0-9]/)) {
    res.render("signup", {
        msg: "Incluye una mayúscula y un número en tu contraseña."
    })
    return
}
```

#APLICACIÓN

También para crear patrones dinámicos en tus rutas:

```
router.get(/^\/(api|rest)\/allItems$/, (req, res) => {
 res.render('view')
})
```

O validar, en el equema de un modelo, la idoneidad de un campo de MongoDB previo a insertarlo en la Base de Datos:

```
const carSchema = new Schema({
 model: String,
 plate: {
   type: String,
   match: /^[0-9]{4}[A-Z]{2}$/
 }
})
```

#APLICACIÓN

/

Indica el inicio y el final de la expresión regular.

Dentro de una expresión regular podemos encontrar tanto **caracteres literales** (letras, números, espacios...) como **caracteres especiales.**

#INTRODUCCIÓN

`i`

El flag `i` que sigue a una expresión regular **la convierte en case insensitive**, lo que omite las mayúsculas y minúsculas en su evaluación.

#FLAGS

\

La barra invertida previa a ciertos caracteres los convierten en un **caracter especial**, habiendo un amplio rango de caracteres especiales que representan infinidad de comportamientos por parte del patrón.

#CARACTERES_ESPECIALES

\d

El caracter **\d** representa un dígito básico.

#CARACTERES_ESPECIALES

{n}

El caracter **{n}** encuentra exactamente **n** ocurrencias del elemento precedente.

Puede conformar también un **rango de ocurrencias**:

{n,}	busca un mínimo de n veces el elemento precedente
{n,m}	busca entre n y m veces el elemento precedente

#CARACTERES_ESPECIALES

[xyz]

El caracter **[xyz]** representa un set o juego: habrá una concurrencia con cualquiera de los caracteres contenidos en el mismo.

Puede conformar también un **juego de ocurrencias**:

[a-z]	busca una minúscula
[A-Z]	busca una mayúscula
[0-9]	busca un número

#CARACTERES_ESPECIALES

/^expr$/

Comenzando la expresión con **/^** y finalizando con **$/** haremos una expresión estricta: el String deberá adaptarse en su totalidad al patrón.

|

El operador | representa en una expresión regular lo mismo que en una expresión condicional: valida el String tanto si cumple el patrón contenido a su izquierda como a su derecha.

#CARACTERES_ESPECIALES

+

El caracter `+` encuentra una o más veces el elemento precedente.

Equivalente a `{n,}`

#CARACTERES_ESPECIALES