**Expresiones regulares**

**Usa el método "test"**

Las expresiones regulares se utilizan en lenguajes de programación para coincidir con partes de cadenas. Creas patrones para ayudarte a hacer esa coincidencia.

Si quieres encontrar la palabra the en la cadena The dog chased the cat, puedes utilizar la siguiente expresión regular: /the/. Ten en cuenta que las comillas no son requeridas dentro de la expresión regular.

JavaScript tiene múltiples formas de usar expresiones regulares. Una forma de probar una expresión regular es usando el método .test(). El método .test() toma la expresión regular, la aplica a una cadena (que se coloca dentro de los paréntesis), y devuelve true o false si tu patrón encuentra algo o no.

**let testStr = "freeCodeCamp";**

**let testRegex = /Code/;**

**testRegex.test(testStr);**

**El método test aquí devuelve true.**

**let myString = "Hello, World!";**

**let myRegex = /Hello/;**

**let result = myRegex.test(myString); // Cambia esta línea**

**Haz coincidir cadenas literales**

En el desafío anterior, buscaste la palabra Hello usando la expresión regular /Hello/. Esa expresión regular buscó una coincidencia literal de la cadena Hello. Aquí hay otro ejemplo donde se busca una coincidencia literal de la cadena Kevin:

**let testStr = "Hello, my name is Kevin.";**

**let testRegex = /Kevin/;**

**testRegex.test(testStr);**

Esta llamada a test devolverá true.

Cualquier otra variante de Kevin no coincidirá. Por ejemplo, la expresión regular /Kevin/ no coincidirá con kevin o KEVIN.

**let wrongRegex = /kevin/;**

**wrongRegex.test(testStr);**

Esta llamada a test devolverá false.

Un futuro desafío también mostrará cómo coincidir esas otras variantes.

**let waldoIsHiding = "Somewhere Waldo is hiding in this text.";**

**let waldoRegex = /Waldo/; // Cambia esta línea**

**let result = waldoRegex.test(waldoIsHiding);**

**Coincide una cadena literal con diferentes posibilidades**

Al utilizar expresiones regulares como /coding/, puedes buscar el patrón coding en otra cadena.

Esto es muy potente al buscar cadenas individuales, pero está limitado a un solo patrón. Puedes buscar múltiples patrones usando el operador alternation o OR: |.

Este operador coincide con los patrones antes o después de él. Por ejemplo, si deseas coincidir con las cadenas yes o no, la expresión regular que quieres es /yes|no/.

También puedes buscar más de dos patrones. Puedes hacer esto añadiendo más patrones con más operadores OR separándolos, como **/yes|no|maybe/.**

**let petString = "James has a pet cat.";**

**let petRegex = /dog|cat|bird|fish/; // Cambia esta línea**

**let result = petRegex.test(petString);**

**Ignora la capitalización al coincidir**

Hasta ahora, has visto expresiones regulares para hacer coincidir cadenas literales. Pero a veces, tal vez quieras hacer coincidir las diferencias de capitalización.

La capitalización (o también llamada capitalización de letra) es la diferencia entre mayúsculas y minúsculas. Ejemplos de mayúsculas son A, B y C. Ejemplos de minúsculas son a, b y c.

Puedes coincidir ambos casos utilizando algo llamado bandera. Existen otras banderas, pero aquí te centrarás en la que ignora la capitalización de las letras, la bandera i. Puedes usarla añadiéndola a la expresión regular. Un ejemplo de uso de esta bandera es /ignorecase/i. Esta expresión regular puede coincidir con las cadenas ignorecase, igNoreCase e IgnoreCase.

**let myString = "freeCodeCamp";**

**let fccRegex = /freeCodeCamp/i; // Cambia esta línea**

**let result = fccRegex.test(myString);**

**Extrae coincidencias**

Hasta ahora, sólo has estado comprobando si un patrón existe o no dentro de una cadena. También puedes extraer las coincidencias encontradas con el método .match().

Para utilizar el método .match(), aplica el método a una cadena y pasa la expresión regular dentro de los paréntesis.

Este es un ejemplo:

**"Hello, World!".match(/Hello/);**

**let ourStr = "Regular expressions";**

**let ourRegex = /expressions/;**

**ourStr.match(ourRegex);**

Aquí el primer match devolverá ["Hello"] y el segundo devolverá ["expressions"].

Ten en cuenta que la sintaxis .match es lo "opuesto" al método .test que has estado utilizando hasta ahora:

**'string'.match(/regex/);**

**/regex/.test('string');**

**let extractStr = "Extract the word 'coding' from this string.";**

**let codingRegex = /coding/; // Cambia esta línea**

**let result = extractStr.match(codingRegex); // Cambia esta línea**

**Encuentra más que la primera coincidencia**

Hasta ahora, sólo has podido extraer o buscar un patrón una vez.

**let testStr = "Repeat, Repeat, Repeat";**

**let ourRegex = /Repeat/;**

**testStr.match(ourRegex);**

Aquí match devolverá ["Repeat"].

Para buscar o extraer un patrón más de una vez, puedes utilizar la bandera g.

**let repeatRegex = /Repeat/g;**

**testStr.match(repeatRegex);**

Y aquí match devuelve el valor ["Repeat", "Repeat", "Repeat"]

**let twinkleStar = "Twinkle, twinkle, little star";**

**let starRegex = /Twinkle/ig; // Cambia esta línea**

**let result = twinkleStar.match(starRegex); // Cambia esta línea**

**Haz coincidir cualquier cosa con el comodín punto**

A veces no conoces (o no necesitas conocer) los caracteres exactos en tus patrones. Pensar en todas las palabras que coincidan, digamos, con una ortografía errónea llevaría mucho tiempo. Afortunadamente, puedes ahorrar tiempo utilizando el carácter de comodín: .

El carácter de comodín . coincidirá con cualquier carácter único. El comodín también es llamado dot y period. Puedes utilizar el carácter de comodín como cualquier otro carácter en la expresión regular. Por ejemplo, si quieres hacer coincidir hug, huh, hut, y hum, puedes usar la la expresión regular /hu./ para que coincida con las cuatro palabras.

**let humStr = "I'll hum a song";**

**let hugStr = "Bear hug";**

**let huRegex = /hu./;**

**huRegex.test(humStr);**

**huRegex.test(hugStr);**

Ambas llamadas a test devolverán true.

**let exampleStr = "Let's have fun with regular expressions!";**

**let unRegex = /.un/; // Cambia esta línea**

**let result = unRegex.test(exampleStr);**

**Haz coincidir un solo carácter con múltiples posibilidades**

Aprendiste cómo hacer coincidir los patrones literales (/literal/) y el carácter de comodín (/./). Estos son los extremos de las expresiones regulares, donde uno encuentra coincidencias exactas y el otro coincide de todo. Hay opciones que representan un equilibrio entre los dos extremos.

Puedes buscar un patrón literal con cierta flexibilidad utilizando las clases de caracteres. Las clases de caracteres te permiten definir un grupo de caracteres que deseas coincidir colocándolos dentro de corchetes ([ y ]).

Por ejemplo, si quieres hacer coincidir bag, big, y bug pero no bog. Puedes crear la expresión regular /b[aiu]g/ para hacer esto. [aiu] es la clase de caracteres que solo coincidirá con los caracteres a, i, o u.

**let bigStr = "big";**

**let bagStr = "bag";**

**let bugStr = "bug";**

**let bogStr = "bog";**

**let bgRegex = /b[aiu]g/;**

**bigStr.match(bgRegex);**

**bagStr.match(bgRegex);**

**bugStr.match(bgRegex);**

**bogStr.match(bgRegex);**

En orden, las cuatro llamadas de match devolverán los valores ["big"], ["bag"], ["bug"], and null.

**let quoteSample = "Beware of bugs in the above code; I have only proved it correct, not tried it.";**

**let vowelRegex = /[aeiou]/ig; // Cambia esta línea**

**let result = quoteSample.match(vowelRegex); // Cambia esta línea**

**Haz coincidir las letras del alfabeto**

Has visto cómo puedes usar los conjuntos de caracteres para especificar un grupo de caracteres a coincidir, pero eso requiere escribir mucho cuando necesitas coincidir con un amplio rango de caracteres (por ejemplo, cada letra en el alfabeto). Afortunadamente, hay una funcionalidad incorporada que hace esto corto y sencillo.

Dentro de un conjunto de caracteres, puedes definir un rango de caracteres a coincidir usando un carácter de guion: -.

Por ejemplo, para hacer coincidir las letras minúsculas desde la a a la e usarías [a-e].

**let catStr = "cat";**

**let batStr = "bat";**

**let matStr = "mat";**

**let bgRegex = /[a-e]at/;**

**catStr.match(bgRegex);**

**batStr.match(bgRegex);**

**matStr.match(bgRegex);**

En orden, las tres llamadas a match devolverán los valores ["cat"], ["bat"] y null.

**let quoteSample = "The quick brown fox jumps over the lazy dog.";**

**let alphabetRegex = /[a-z]/ig; // Cambia esta línea**

**let result = quoteSample.match(alphabetRegex); // Cambia esta línea**

**Haz coincidir los números y las letras del alfabeto**

Usar el guión (-) para coincidir con un rango de caracteres no está limitado a letras. También funciona para hacer coincidir un rango de números.

Por ejemplo, /[0-5]/ coincide con cualquier número entre 0 y 5, incluyendo 0 y 5.

Además, es posible combinar un rango de letras y números en un único conjunto de caracteres.

**let jennyStr = "Jenny8675309";**

**let myRegex = /[a-z0-9]/ig;**

**jennyStr.match(myRegex);**

**let quoteSample = "Blueberry 3.141592653s are delicious.";**

**let myRegex = /[2-6h-s]/ig; // Cambia esta línea**

**let result = quoteSample.match(myRegex); // Cambia esta línea**

**Haz coincidir caracteres individuales no especificados**

Hasta ahora, has creado un conjunto de caracteres que deseas coincidir, pero también podrías crear un conjunto de caracteres que no desees coincidir. Este tipo de conjuntos de caracteres se llaman conjuntos de caracteres negados.

Para crear un conjunto de caracteres negados colocas un carácter de intercalación (^) después del corchete de apertura y antes de los caracteres que no quieres coincidir.

Por ejemplo, **/[^aeiou]/gi** coincide con todos los caracteres que no son una vocal. Ten en cuenta que caracteres como ., !, [, @, / y el espacio en blanco coinciden; el conjunto de caracteres de vocal negados sólo excluye los caracteres de vocal.

**let quoteSample = "3 blind mice.";**

**let myRegex = /[^0-9aeiou]/ig; // Cambia esta línea**

**let result = quoteSample.match(myRegex); // Cambia esta línea**

**Haz coincidir caracteres que aparecen una o más veces**

A veces, es necesario coincidir con un carácter (o grupo de caracteres) que aparezca una o más veces seguidas. Esto significa que aparece al menos una vez, y puede repetirse.

Puedes usar el carácter + para comprobar si es así. Recuerda, el carácter o patrón debe estar presente consecutivamente. Es decir, el carácter tiene que repetirse uno tras otro.

Por ejemplo, /a+/g encontraría una coincidencia en abc y regresaría ["a"]. Debido al +, también encontraría una sola coincidencia en aabc y regresaría ["aa"].

Si en su lugar estuvieras comprobando la cadena abab, se encontrarían dos coincidencias y regresaría ["a", "a"] porque los caracteres a no están en fila; hay una b entre ellos. Finalmente, dado que no hay una a en la cadena bcd, no se encontraría una coincidencia.

**let difficultSpelling = "Mississippi";**

**let myRegex = /s+/g; // Cambia esta línea**

**let result = difficultSpelling.match(myRegex);**

**Haz coincidir caracteres que aparecen cero o más veces**

El último desafío utilizó el signo más + para buscar caracteres que aparecen una o más veces. También hay una opción para hacer coincidir caracteres que aparecen cero o más veces.

El carácter que hace esto es el asterisco o la estrella: \*.

**let soccerWord = "gooooooooal!";**

**let gPhrase = "gut feeling";**

**let oPhrase = "over the moon";**

**let goRegex = /go\*/;**

**soccerWord.match(goRegex);**

**gPhrase.match(goRegex);**

**oPhrase.match(goRegex);**

En orden, los tres match devolverán los valores ["goooooooo"], ["g"], y null.

**let chewieQuote = "Aaaaaaaaaaaaaaaarrrgh!";**

**let chewieRegex = /Aa\*/; // Change this line**

**let result = chewieQuote.match(chewieRegex);**

**Encuentra caracteres con una coincidencia perezosa**

En las expresiones regulares, una coincidencia codiciosa encuentra la parte más larga posible de una cadena que se ajusta al patrón de la expresión regular y la devuelve como una coincidencia. La alternativa es llamada coincidencia perezosa, la cual encuentra la parte más pequeña posible de la cadena que satisface el patrón de la expresión regular.

Puedes aplicar la expresión regular /t[a-z]\*i/ a la cadena "titanic". Esta expresión regular es básicamente un patrón que comienza con t, termina con i, y tiene algunas letras intermedias.

Las expresiones regulares son por defecto codiciosas, por lo que la coincidencia devolvería ["titani"]. Encuentra la sub-cadena más grande posible que se ajusta al patrón.

Sin embargo, puedes usar el carácter ? para cambiarla a una coincidencia perezosa. "titanic" al coincidir con la expresión regular ajustada de /t[a-z]\*?i/ devuelve ["ti"].

Nota: Se debe evitar analizar HTML con expresiones regulares, pero coincidir patrones con una cadena HTML utilizando expresiones regulares está completamente bien.

**let text = "<h1>Winter is coming</h1>";**

**let myRegex = /<.\*?>/; // Cambia esta línea**

**let result = text.match(myRegex);**

**Encuentra uno o más criminales en una cacería**

Es hora de hacer una pausa y probar tus nuevas habilidades para escribir expresiones regulares. Un grupo de criminales se han escapado de la cárcel, pero no sabes cuántos. Sin embargo, sabes que permanecen unidos cuando están alrededor de otras personas. Eres responsable de encontrar a todos los criminales a la vez.

Este es un ejemplo para revisar cómo hacer esto:

La expresión regular /z+/ coincide con la letra z cuando aparece una o más veces seguidas. Encontrará coincidencias en las siguientes cadenas:

**"z"**

**"zzzzzz"**

**"ABCzzzz"**

**"zzzzABC"**

**"abczzzzzzzzzzzzzzzzzzzzzabc"**

Pero no encuentra coincidencias en las siguientes cadenas, ya que no hay letras z:

**""**

**"ABC"**

**"abcabc"**

**let reCriminals = /C+/; // Change this line**

**Haz coincidir patrones de cadena de inicio**

Los desafíos anteriores demostraron que las expresiones regulares pueden ser utilizadas para buscar una serie de coincidencias. También se utilizan para buscar patrones en posiciones específicas en cadenas.

En un desafío anterior, usaste el carácter caret (^) dentro de un conjunto de caracteres para crear un conjunto de caracteres en la forma [^thingsThatWillNotBeMatched]. Fuera de un conjunto de caracteres, el caret es utilizado para buscar patrones al principio de las cadenas.

**let firstString = "Ricky is first and can be found.";**

**let firstRegex = /^Ricky/;**

**firstRegex.test(firstString);**

**let notFirst = "You can't find Ricky now.";**

**firstRegex.test(notFirst);**

La primera llamada test devolverá true, mientras que la segunda retornara false.

**let rickyAndCal = "Cal and Ricky both like racing.";**

**let calRegex = /^Cal/; // Cambia esta línea**

**let result = calRegex.test(rickyAndCal);**

**Haz coincidir patrones de cadena final**

En el último desafío, aprendiste a usar el carácter de intercalación para buscar patrones al inicio de las cadenas. También hay una manera de buscar patrones al final de las cadenas.

Puedes buscar el final de las cadenas usando el carácter del signo de dólar $ al final de la expresión regular.

**let theEnding = "This is a never ending story";**

**let storyRegex = /story$/;**

**storyRegex.test(theEnding);**

**let noEnding = "Sometimes a story will have to end";**

**storyRegex.test(noEnding);**

La primera llamada a test devuelve true, mientras que la segunda retorna false.

**let caboose = "The last car on a train is the caboose";**

**let lastRegex = /caboose$/; // Cambia esta línea**

**let result = lastRegex.test(caboose);**

**Coincide todas las letras y números**

Usando clases de caracteres, pudiste buscar todas las letras del alfabeto con [a-z]. Este tipo de clase de caracteres es tan común que existe un atajo para él, aunque también incluye algunos caracteres adicionales.

La clase de caracteres más cercana en JavaScript para coincidir con el alfabeto es \w. Este atajo equivale a [A-Za-z0-9\_]. Esta clase de caracteres coincide con letras mayúsculas y minúsculas más números. Ten en cuenta que esta clase de caracteres también incluye el carácter de guión bajo (\_).

**let longHand = /[A-Za-z0-9\_]+/;**

**let shortHand = /\w+/;**

**let numbers = "42";**

**let varNames = "important\_var";**

**longHand.test(numbers);**

**shortHand.test(numbers);**

**longHand.test(varNames);**

**shortHand.test(varNames);**

**Las cuatro llamadas a test devolverán true.**

Estos atajos de clases de caracteres también son conocidos como clases de caracteres abreviados.

**let quoteSample = "The five boxing wizards jump quickly.";**

**let alphabetRegexV2 = /\w/g; // Cambia esta línea**

**let result = quoteSample.match(alphabetRegexV2).length;**

**Haz coincidir todo menos letras y números**

Has aprendido que puedes usar un atajo para emparejar alfanuméricos [A-Za-z0-9\_] usando \w. Un patrón natural que tal vez quieras buscar es lo contrario a la alfanumérica.

Puedes buscar lo contrario de \w con \W. Ten en cuenta, el patrón contrario usa letra mayúscula. Este atajo es lo mismo que [^A-Za-z0-9\_].

**let shortHand = /\W/;**

**let numbers = "42%";**

**let sentence = "Coding!";**

**numbers.match(shortHand);**

**sentence.match(shortHand);**

El primer match devuelve el valor ["%"] y el segundo devuelve ["!"]

**let quoteSample = "The five boxing wizards jump quickly.";**

**let nonAlphabetRegex = /\W/g; // Cambia esta línea**

**let result = quoteSample.match(nonAlphabetRegex).length;**

**Coincide con todos los números**

Has aprendido atajos para patrones de cadenas comunes como los alfanuméricos. Otro patrón común es buscar solo dígitos o números.

El atajo para buscar caracteres de dígitos es \d, con una d minúscula. Esto es igual a la clase de caracteres [0-9], la cual busca un solo carácter de cualquier número entre cero y nueve.

**let movieName = "2001: A Space Odyssey";**

**let numRegex = /\d/g; // Cambia esta línea**

**let result = movieName.match(numRegex).length;**

**Coincide con todos los caracteres no numéricos**

El último desafío mostró cómo buscar dígitos usando el atajo \d con una d minúscula. También puedes buscar caracteres que no sean dígitos usando un atajo similar que utilice una D mayúscula en su lugar.

El atajo para buscar caracteres que no sean dígitos es \D. Esto es igual a la clase de caracteres [^0-9], el cual busca un único carácter que no sea un número entre cero y nueve.

**let movieName = "2001: A Space Odyssey";**

**let noNumRegex = /\D/g; // Cambia esta línea**

**let result = movieName.match(noNumRegex).length;**

**estringe posibles nombres de usuario**

Los nombres de usuario se utilizan en todas partes en Internet. Son los que dan a los usuarios una identidad única en tus sitios favoritos.

Se necesita comprobar todos los nombres de usuario en una base de datos. Estas son algunas reglas simples que los usuarios deben seguir al crear su nombre de usuario.

Los nombres de usuario sólo pueden utilizar caracteres alfanuméricos.

Los únicos números del nombre de usuario tienen que estar al final. Puede tener un cero o más al final. El nombre de usuario no puede iniciar con un número.

Las letras del nombre de usuario pueden ser minúsculas y mayúsculas.

Los nombres de usuario deben tener al menos dos caracteres. Un nombre de usuario de dos caracteres sólo puede utilizar letras del alfabeto como caracteres.

**let username = "JackOfAllTrades";**

**let userCheck = /^[a-z]([0-9]{2,}|[a-z]+\d\*)$/i; // Cambia esta línea**

**let result = userCheck.test(username);**

**Código Explicación**

1. **^- inicio de entrada**
2. **[a-z]- el primer carácter es una letra**
3. **[0-9]{2,}- termina con dos o más números**
4. **|- o**
5. **[a-z]+- tiene una o más letras al lado**
6. **\d\*- y termina con cero o más números**
7. **$- fin de entrada**
8. **i- ignorar caso de entrada**

**Haz coincidir espacios en blanco**

Los desafíos por el momento han cubierto las letras que coinciden con el alfabeto y los números. También puedes hacer que coincidan los espacios en blanco o los espacios entre las letras.

Puedes buscar los espacios en blanco usando \s que es una s minúscula. Este patrón no solo coincide con los espacios en blanco, también con los caracteres de retorno de carro, tabulaciones, alimentación de formulario y saltos de línea. Puedes pensar que es similar a las clases de caracteres [ \r\t\f\n\v].

**let whiteSpace = "Whitespace. Whitespace everywhere!"**

**let spaceRegex = /\s/g;**

**whiteSpace.match(spaceRegex);**

Esta llamada a match debe devolver [" ", " "].

**let sample = "Whitespace is important in separating words";**

**let countWhiteSpace = /\s/g; // Cambia esta línea**

**let result = sample.match(countWhiteSpace);**

**Haz coincidir caracteres que no sean espacios en blanco**

Aprendiste a buscar espacios en blanco usando \s, con una s en minúscula. También puedes buscar todo excepto los espacios en blanco.

Busca caracteres que no sean espacios en blanco usando \S, la cual es una s mayúscula. Este patrón no coincidirá con los caracteres de espacios en blanco, retorno de carro, tabulaciones, alimentación de formulario y saltos de línea. Puedes pensar que es similar a la clase de caracteres [^ \r\t\f\n\v].

**let whiteSpace = "Whitespace. Whitespace everywhere!"**

**let nonSpaceRegex = /\S/g;**

**whiteSpace.match(nonSpaceRegex).length;**

El valor devuelto por el método .length sería 32.

**let sample = "Whitespace is important in separating words";**

**let countNonWhiteSpace = /\S/g; // Cambia esta línea**

**let result = sample.match(countNonWhiteSpace);**

**Especifica el menor y mayor número de coincidencias**

Recuerda que se utiliza el signo más + para buscar uno o más caracteres y el asterisco \* para buscar cero o más caracteres. Esto es conveniente, pero a veces quieres coincidir con cierta gama de patrones.

Puedes especificar el número inferior y superior de patrones utilizando especificadores de cantidad. Para los especificadores de cantidad utilizamos llaves ({ y }). Pon dos números entre las llaves - para el número inferior y superior de patrones.

Por ejemplo, para que coincida con la letra a si aparece entre 3 y 5 veces en la cadena ah, la expresión regular debe ser /a{3,5}h/.

**let A4 = "aaaah";**

**let A2 = "aah";**

**let multipleA = /a{3,5}h/;**

**multipleA.test(A4);**

**multipleA.test(A2);**

La primera llamada a test devuelve true, mientras que la segunda devuelve false.

**let ohStr = "Ohhh no";**

**let ohRegex = /Oh{3,6}\sno/; // Cambia esta línea**

**let result = ohRegex.test(ohStr);**

**Especifica solo el menor número de coincidencias**

Puedes especificar el número inferior y superior de patrones mediante especificadores de cantidad utilizando llaves. A veces sólo se quiere especificar el número inferior de patrones sin tener un límite superior.

Para especificar sólo el número inferior de patrones, mantén el primer número seguido de una coma.

Por ejemplo, para hacer coincidir solo con la cadena hah cuando la letra a aparezca al menos 3 veces, la expresión regular sería /ha{3,}h/.

**let A4 = "haaaah";**

**let A2 = "haah";**

**let A100 = "h" + "a".repeat(100) + "h";**

**let multipleA = /ha{3,}h/;**

**multipleA.test(A4);**

**multipleA.test(A2);**

**multipleA.test(A100);**

En orden, las tres llamadas a test devuelven true, false y true.

**let haStr = "Hazzzzah";**

**let haRegex = /Haz{4,}ah/; // Cambia esta línea**

**let result = haRegex.test(haStr);**

**Especifica el número exacto de coincidencias**

Puedes especificar el número inferior y superior de patrones mediante especificadores de cantidad utilizando llaves. A veces solo quieres un número específico de coincidencias.

Para especificar un cierto número de patrones, simplemente pon ese número entre corchetes.

Por ejemplo, para que coincida con la palabra hah solo con la letra a 3 veces, tu expresión regular sera /ha{3}h/.

**let A4 = "haaaah";**

**let A3 = "haaah";**

**let A100 = "h" + "a".repeat(100) + "h";**

**let multipleHA = /ha{3}h/;**

**multipleHA.test(A4);**

**multipleHA.test(A3);**

**multipleHA.test(A100);**

En orden, las tres llamadas a test devuelven false, true y false.

**let timStr = "Timmmmber";**

**let timRegex = /Tim{4}ber/; // Cambia esta línea**

**let result = timRegex.test(timStr);**

**Comprueba todos o ninguno**

A veces los patrones que quieres buscar pueden tener partes que pueden o no existir. Sin embargo, podría ser importante buscarlos de todos maneras.

Puedes especificar la posible existencia de un elemento con un signo de interrogación, ?. Esto comprueba cero o uno de los elementos precedentes. Puedes pensar que este símbolo dice que el elemento anterior es opcional.

Por ejemplo, hay ligeras diferencias en inglés americano y británico y puedes usar el signo de interrogación para coincidir con ambas ortografías.

**let american = "color";**

**let british = "colour";**

**let rainbowRegex= /colou?r/;**

**rainbowRegex.test(american);**

**rainbowRegex.test(british);**

Ambos usos del método test devolverán true.

**let favWord = "favorite";**

**let favRegex = /favou?rite/; // Cambia esta línea**

**let result = favRegex.test(favWord);**

**Lookahead positivo y negativo**

Los lookaheads son patrones que le indican a JavaScript que busque por anticipado en tu cadena para verificar patrones más adelante. Esto puede ser útil cuando deseas buscar varios patrones sobre la misma cadena.

Hay dos tipos de lookaheads: lookahead positivo y lookahead negativo.

Un lookahead positivo buscará para asegurarse de que el elemento en el patrón de búsqueda este allí, pero en realidad no lo coincidirá. Un lookahead positivo se usa como (?=...) donde el ... es la parte requerida que no coincide.

Por otro lado, un lookahead negativo buscará para asegurarse de que el elemento en el patrón de búsqueda no este allí. Un lookahead negativo se usa como (?!...) donde el ... es el patrón que no quieres que esté allí. El resto del patrón se devuelve si la parte de lookahead negativo no está presente.

Los lookaheads son un poco confusos, pero algunos ejemplos ayudarán.

**let quit = "qu";**

**let noquit = "qt";**

**let quRegex= /q(?=u)/;**

**let qRegex = /q(?!u)/;**

**quit.match(quRegex);**

**noquit.match(qRegex);**

Ambas llamadas a match devolverán ["q"].

Un uso más práctico de lookaheads es comprobar dos o más patrones en una cadena. Aquí hay un verificador de contraseñas (ingenuamente) simple que busca entre 3 y 6 caracteres y al menos un número:

**let password = "abc123";**

**let checkPass = /(?=\w{3,6})(?=\D\*\d)/;**

**checkPass.test(password);**

**Lookahead positivo y negativo**

Los lookaheads son patrones que le indican a JavaScript que busque por anticipado en tu cadena para verificar patrones más adelante. Esto puede ser útil cuando deseas buscar varios patrones sobre la misma cadena.

Hay dos tipos de lookaheads: lookahead positivo y lookahead negativo.

Un lookahead positivo buscará para asegurarse de que el elemento en el patrón de búsqueda este allí, pero en realidad no lo coincidirá. Un lookahead positivo se usa como (?=...) donde el ... es la parte requerida que no coincide.

Por otro lado, un lookahead negativo buscará para asegurarse de que el elemento en el patrón de búsqueda no este allí. Un lookahead negativo se usa como (?!...) donde el ... es el patrón que no quieres que esté allí. El resto del patrón se devuelve si la parte de lookahead negativo no está presente.

Los lookaheads son un poco confusos, pero algunos ejemplos ayudarán.

**let quit = "qu";**

**let noquit = "qt";**

**let quRegex= /q(?=u)/;**

**let qRegex = /q(?!u)/;**

**quit.match(quRegex);**

**noquit.match(qRegex);**

Ambas llamadas a match devolverán ["q"].

Un uso más práctico de lookaheads es comprobar dos o más patrones en una cadena. Aquí hay un verificador de contraseñas (ingenuamente) simple que busca entre 3 y 6 caracteres y al menos un número:

**let password = "abc123";**

**let checkPass = /(?=\w{3,6})(?=\D\*\d)/;**

**checkPass.test(password);**

Utiliza los lookaheads en el pwRegex para que coincida con las contraseñas que tengan más de 5 caracteres y dos dígitos consecutivos.

**let sampleWord = "astronaut";**

**let pwRegex = /(?=\w{6})(?=\w\*\d{2})/; // Cambia esta línea**

**let result = pwRegex.test(sampleWord);**

**Comprueba agrupaciones mixtas de caracteres**

A veces queremos comprobar grupos de caracteres utilizando una expresión regular y para conseguirlo usamos paréntesis ().

Si deseas encontrar Penguin o Pumpkin en una cadena, puedes usar la siguiente expresión regular: /P(engu|umpk)in/g

Luego, comprueba si los grupos de cadena deseados están en la cadena de prueba usando el método test().

**let testStr = "Pumpkin";**

**let testRegex = /P(engu|umpk)in/;**

**testRegex.test(testStr);**

El método test aquí devolverá true.

**let myString = "Eleanor Roosevelt";**

**let myRegex = /(Franklin|Eleanor) (([A-Z]\.?|[A-Z][a-z]+) )?Roosevelt/; // Cambia esta línea**

**let result = myRegex.test(myString); // Cambia esta línea**

**// Después de pasar el desafío experimenta con myString y observa cómo funciona la agrupación**

**Reutiliza patrones usando grupos de captura**

Digamos que quieres hacer coincidir una palabra que aparece varias veces como la siguiente.

let repeatStr = "row row row your boat";

Podrías usar /row row row/, pero ¿qué pasa si no conoces la palabra específica que se repite? Los grupos de captura pueden utilizarse para encontrar subcadenas repetidas.

Los grupos de captura se construyen encerrando entre paréntesis el patrón de expresión regular a capturar. En este caso, el objetivo es capturar una palabra formada por caracteres alfanuméricos, por lo que el grupo de captura será \w+ encerrado entre paréntesis: /(\w+)/.

La subcadena que coincide con el grupo se guarda en una "variable" temporal, a la que se puede acceder dentro de la misma expresión regular utilizando una barra invertida y el número del grupo de captura (por ejemplo, \1). Los grupos de captura se numeran automáticamente por la posición de sus paréntesis de apertura (de izquierda a derecha), empezando por el 1.

El siguiente ejemplo encuentra cualquier palabra que aparezca tres veces separada por un espacio:

**let repeatRegex = /(\w+) \1 \1/;**

**repeatRegex.test(repeatStr); // Returns true**

**repeatStr.match(repeatRegex); // Returns ["row row row", "row"]**

El uso del método .match() en una cadena devolverá un arreglo con la subcadena coincidente, junto con sus grupos capturados.

**let repeatNum = "42 42 42";**

**let reRegex = /^(\d+) \1 \1(?!.)/; // Cambia esta línea**

**let result = reRegex.test(repeatNum);**

**Usa grupos de captura para buscar y reemplazar**

La búsqueda es útil. Sin embargo, puedes hacer que la búsqueda sea aún más poderosa si también cambias (o reemplazas) el texto con el que coincide.

Puedes buscar y reemplazar texto en una cadena usando .replace() en una cadena. Las entradas para .replace() son primero el patrón de expresiones regulares que deseas buscar. El segundo parámetro es la cadena para reemplazar la coincidencia o una función para hacer algo.

**let wrongText = "The sky is silver.";**

**let silverRegex = /silver/;**

**wrongText.replace(silverRegex, "blue");**

La llamada replace devolverá la cadena The sky is blue..

También puedes acceder a grupos de captura en la cadena de reemplazo con signos de dólar. ($).

**"Code Camp".replace(/(\w+)\s(\w+)/, '$2 $1');**

La llamada replace devolverá la cadena Camp Code.

**let str = "one two three";**

**let fixRegex = /(\w+)\s(\w+)\s(\w+)/; // Cambia esta línea**

**let replaceText = "$3 $2 $1"; // Cambia esta línea**

**let result = str.replace(fixRegex, replaceText);**

**Elimina espacio en blanco del inicio y final**

A veces no se quieren caracteres en espacios en blanco alrededor de las cadenas, pero están ahí. El proceso típico de las cadenas de texto es eliminar el espacio en blanco al inicio y al final del mismo.

**let hello = " Hello, World! ";**

**let wsRegex = /^\s+|\s+$/g; // Cambia esta línea**

**let result = hello.replace(wsRegex, ''); // Cambia esta línea**