Expresiones Regulares.

Las expresiones regulares son patrones que ayudan a los programadores a encontrar, buscar y reemplazar texto. Son muy potentes pero pueden ser difíciles de leer porque usan caracteres especiales para hacer coincidencias más complejas y flexibles.

1.- Usa 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 myString = "Hello, World!"; let myRegex = /Hello/; let result = myRegex.test(myString);	Aplica la expresión regular myRegex en la cadena myString usando el método .test().

2.- Haz coincidir cadenas literales:

let waldoIsHiding = "Somewhere Waldo is hiding in this text."; let waldoRegex = /Waldo/; let result = waldoRegex.test(waldoIsHiding);	Completa la expresión regular waldoRegex para encontrar “Waldo” en la cadena waldoIsHiding con una coincidencia literal.

3.- Coincide una cadena literal con diferentes posibilidades:

Puedes buscar múltiples patrones usando el operador alternation o OR [|]. Este operador coincide con los patrones antes y después de él. Por ejemplo si deseas coincidir con las cadenas yes or no, la expresión regular que quieres es [/yes|no/]. También puedes buscar más de dos patrones [/yes|no|maybe/].

let petString = "James has a pet cat."; let petRegex = /dog|cat|bird|fish/; let result = petRegex.test(petString);	Completa la expresión regular petRegex para que coincida con las mascotas dog, cat, bird o fish.

4.- Ignora la capitalización al coincidir:

En caso de querer hacer coincidir cadenas con diferente capitalización utilizaremos algo llamado banderas. La bandera i ignora la capitalización. [/ignorecase/i].

let myString = "freeCodeCamp";
let fccRegex = /freecodecamp/i;
let result = fccRegex.test(myString);
	Escribe una expresión regular fccRegex para que coincida con freecodecamp sin importar la capitalización.

5.- Extrae coincidencias:

Hasta ahora sólo hemos comprobado si un patrón existe o no. Con el método .match() podemos extraer coincidencias, aplicándolo a una cadena y pasando expresiones regulares dentro de los paréntesis.

let extractStr = "Extract the word 'coding' from this string."; let codingRegex = /coding/; let result = extractStr.match(codingRegex);	Aplica el método .match() para extraer una cadena coding.

6.- Encuentra más que la primera coincidencia:

Para buscar o extraer un patrón más de una vez, puedes utilizar la bandera g. La bandera g es global, por lo que extraerá todas las cadenas que coincidan.

let twinkleStar = "Twinkle, twinkle, little star";
let starRegex = /twinkle/gi;
let result = twinkleStar.match(starRegex);
	Utilizando la expresión regular starRegex, encuentra y extrae ambas palabras Twinkle de la cadena twinkleStar.

7.- Haz coincidir cualquier cosa con el comodín punto.

El carácter comodín punto [.] coincidirá con cualquier carácter único. Este comodín también es llamado dot y period. Por ejemplo si quieres hacer coincidir hug, huh, hut y hum, puedes usar la expresión regular /hu./ para que coincidan las cuatro palabras.

let exampleStr = "Let's have fun with regular expressions!"; let unRegex = /.un/; let result = unRegex.test(exampleStr);	Completa la expresión regular unRegex para que coincida con las cadenas run, sun, fun, pun, nun y bun. Tu expresión regular debe usar el carácter comodín.

8.- Haz coincidir un solo carácter con múltiples posiblidades:

Las clases de carácteres te permiten definir un grupo de caracteres que deseas coincidir colocándolos entre corchetes []. Por ejemplo, si quieres hacer coincidir big, bag y bug, pero no bog, puedes hacerlo con [/b[aiu]g/]. Sólo coincidirá con los caracteres a, i, u.

let quoteSample = "Beware of bugs in the above code; I have only proved it correct, not tried it.";
let vowelRegex = /[aeiou]/gi;
let result = quoteSample.match(vowelRegex);
	Usa una clase de caracteres con las vocales (a, e, i, o, u) en tu expresión regular vowelRegex para encontrar todas las vocales en quoteSample. Asegurate de hacer coincidir tanto mayúsculas como minúsculas.

9.- Haz coincidir las letras del alfabeto:

Dentro de un conjunto de caracteres, puedes definir un rango de caracteres a coincidir usando un carácter de guión (-).

let quoteSample = "The quick brown fox jumps over the lazy dog.";
let alphabetRegex = /[a-z]/gi;
let result = quoteSample.match(alphabetRegex);
	Haz coincidir todas las letras en la cadena quoteSample. Asegurate de hacer coincidir tanto mayúsculas como minúsculas.

10.- Haz coincidir los números y las letras del alfabeto:

let quoteSample = "Blueberry 3.141592653s are delicious.";
let myRegex = /[h-s2-6]/gi;
let result = quoteSample.match(myRegex);
	Crea una sola expresión regular que coincida con un rango de letras entre h y s, y de números entre 2 y 6. Recuerda incluir las banderas apropiadas.

11.- Haz coincidir caracteres individuales no especificados:

Las expresiones regulares también nos permiten crear un conjunto de caracteres que no deseemos coincidir, estos reciben el nombre de caracteres negados. Para crear grupos de caracteres negados se coloca un carácter acento circunflejo [^] después del corchete de apertura. 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]/gi;
let result = quoteSample.match(myRegex);
	Crea una sola expresión regular que coincida con todos los caracteres que no son un número o una vocal. Recuerda incluir las banderas apropiadas.

12.- Haz coincidir caracteres que aparecen una o más veces:

Si quieres coincidir un carácter que aparezca una o más veces seguidas, puedes usar +.

let difficultSpelling = "Mississippi";
let myRegex = /s+/g;
let result = difficultSpelling.match(myRegex);
	Quieres encontrar coincidencias cuando la letra s ocurre una o más veces en Mississippi. Escribe una expresión regular que utilice el signo +.

13.- Haz coincidir caracteres que aparecen cero o más veces:

Para hacer coincidir carácteres que coinciden cero o más veces utilizamos (*) de la misma forma que utilizamos (+).

let chewieRegex = /Aa*/; let result = chewieQuote.match(chewieRegex);	Para este desafío, chewieQuote ha sido inicializada entre bastidores con la cadena Aaaaaaaaaaarrrrgh!. Crea una expresión regular chewieRegex que utilice el caracter * para encontrar el carácter A seguido por cero o más a en chewieQuote.

14.- Encuentra caracteres con 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 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: devolverá [“titani”]. Sin embargo, puedes usar el carácter ? para cambiarla a una coincidencia perezosa. Al coincidir con la expresión regular /t[a-z]*?i/ devuelve [“ti”].

let text = " Winter is coming ";
let myRegex = /<.*?>/;
let result = text.match(myRegex);
console.log(result);
	Corrige la expresión regular /<.*>/ para que devuelva la etiqueta HTML <h1> y no el texto "<h1>Winter is coming</h1>". Recuerda que el comodín . en una expresión regular coincide con cualquier carácter.

15.- Encuentra uno o más criminales en una cadena:

let reCriminals = /C+/g;	Escribe una expresión regular codiciosa que encuentre uno o más criminales dentro de un grupo de personas. Un criminal está representado por la letra mayúscula C.

16.- Haz coincidir patrones de cadena de inicio:

Las expresiones regulares pueden utilizarse para buscar patrones en posiciones específicas en cadenas. Anteriormente vimos que el acento circunflejo (^) puede usarse dentro de un conjunto de caracteres para excluir. Por otra parte, si lo usamos fuera de un conjunto su función será buscar patrones al principio de las cadenas.

let rickyAndCal = "Cal and Ricky both like racing."; let calRegex = /^Cal/; let result = calRegex.test(rickyAndCal);	Usa el carácter ^ en una expresión para buscar Cal solo al principio de la cadena rickyAndCal.

17.- Haz coincidir patrones de cadena final:

Puedes buscar patrones usando el carácter ($) al final de una expresión regular.

let caboose = "The last car on a train is the caboose"; let lastRegex = /caboose$/; let result = lastRegex.test(caboose);	Usa el carácter ($) para coincidir la cadena caboose al final de caboose.

18.- Coincide todas las letras y números:

Un atajo para hacer coincidir todo el abecedario, con los números y el carácter especial _ es [\w] que equivaldría a [A-Za-z0-9_].

let quoteSample ="The five boxing wizards jump quickly."; let alphabetRegexV2 = /\w/g; let result =quoteSample.match(alphabetRegexV2).length;	Utiliza la clase de caracteres abreviada \w para contar el número de caracteres alfanuméricos en varias cadenas.

19.- Haz coincidir todo menos letras y números:

let quoteSample ="The five boxing wizards jump quickly."; let nonAlphabetRegex = /\W/g; let result =quoteSample.match(nonAlphabetRegex).length;	Usa la clase de caracteres abreviados \W para contar el número de caracteres no alfanuméricos entre varias cadenas.

20.- Coincide con todos los números:

let movieName = "2001: A Space Odyssey"; let numRegex = /\d/g; let result = movieName.match(numRegex).length;	Usa la clase de caracteres abreviada \d para contar cuántos dígitos hay en los títulos de las películas. Los números escritos (“seis” en lugar de 6) no cuentan. \d equivale a [0-9]

21.- Coincide con todos los caracteres no numéricos:

let movieName = "2001: A Space Odyssey"; let noNumRegex = /\D/g; let result = movieName.match(noNumRegex).length;	Usa la clase de caracteres abreviada \D para contar cuántos caracteres que no sean dígitos hay en los títulos de las películas. \D equivale a [^0-9]

22.- Restringe posibles nombres de usuario:

let username = "JackOfAllTrades"; let userCheck = /^[a-z][a-z]+\d*$\|^[a-z]\d\d+$/i; let result = userCheck.test(username);	Cambia la expresión regular userCheck para que se ajuste a las restricciones indicadas anteriormente.

23.- Haz coincidir espacios en blanco:

Puedes buscar los espacios en blanco usando \s. Este patrón no sólo coincide con los espacios en blanco, también con los caracteres de retorno de carro, tabulaciones, alimentación de formulario y salto de línea [\r\t\f\n\v].

let sample = "Whitespace is important in separating words"; let countWhiteSpace = /\s/g; let result = sample.match(countWhiteSpace);	Cambia la expresión regular countWhiteSpace para buscar múltiples caracteres de espacio en blanco en una cadena.

24.- Haz coincidir caracteres que no sean espacios en blanco:

let sample = "Whitespace is important in separating words"; let countNonWhiteSpace = /\S/g; let result = sample.match(countNonWhiteSpace);	Cambia la expresión regular countNonWhiteSpace para buscar varios caracteres que no sean espacios en blanco con \S.

25.- Especifica el menor y mayor número de coincidencias:

Puedes especificar el número inferior y superior de patrones a coincidir utilizando especificadores de cantidad {}. Por ejemplo, para que coincida con la letra a si aparece entre 3 y 5 veces en la cadena “ah” sería /a{3,5}h/.

let ohStr = "Ohhh no"; let ohRegex = /Oh{3,6}\sno/; let result = ohRegex.test(ohStr);	Modifica la expresión regular ohRegex para que coincida con toda la frase Oh no solo cuando tenga entre 3 y 6 letras h.

26.- Especifica solo el menor número de coincidencias:

let haStr = "Hazzzzah"; let haRegex = /Haz{4,}ah/; let result = haRegex.test(haStr);	Modifica la expresión regular haRegex para coincidir con la palabra Hazzah sólo cuando ésta tenga cuatro o más z.

27.- Especifica el número exacto de coincidencias:

let timStr = "Timmmmber"; let timRegex = /Tim{4}ber/; let result = timRegex.test(timStr);	Modifica la expresión regular timRegex para hacer coincidir con la palabra Timber solo cuando esta tiene cuatro letras m.

28.- Comprueba todos o ninguno:

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.

let favWord = "favorite"; let favRegex = /favou?rite/; let result = favRegex.test(favWord);	Cambia la expresión regular favRegex para que coincida tanto la versión del inglés americano (favorite) como la del británico (favourite).

29.- Lookahead positivo y negativo:

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

Hay dos tipos de lookahead: positivo y negativo. El primero buscará para asegurarse de que el elemento en el patrón de búsqueda esté allí, pero no lo coincidirá. Un lookahead positivo se usa como (?=...) donde … es la parte requerida que no coincide. Por otra parte el lookahead negativo buscará para asegurarse de que el elemento en el patrón de búsqueda no esté allí. Se usa como (?!...) donde el … es el patrón que no quieres que esté allí. El resto del patrón se devuelve si la parte del lookahead negativo no está presente.

El uso más práctico del lookahead es comprobar dos o más patrones en una cadena.

let sampleWord = "astronaut"; let pwRegex = /(?=\w{6})(?=\w*\d{2})/; let result = pwRegex.test(sampleWord);	Utiliza los lookaheads en el pwRegex para que coincidan las contraseñas que tengan más de 5 caracteres y dos dígitos consecutivos.

30.- 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;

let myString = "Eleanor Roosevelt"; let myRegex = /(Franklin|Eleanor) (([A-Z]\.?|[A-Z][a-z]+))?Roosevelt/; let result = myRegex.test(myString);	Corrige la expresión regular para que compruebe los nombres, con sensibilidad a mayus y haciendo concesiones para los apellidos. Luego, corrige el código para que la expresión regular se compruebe con myString y devuelva true o false.

31.- Reutiliza patrones usando grupos de captura:

Los grupos de captura pueden utilizarse para encontrar subcadenas repetidas. Se construyen encerrando entre paréntesis el patrón de la expresión regular a capturar /(\w+)/. La subcadena que coincide con el grupo se guarda en una variable temporal, a la que se puede acceder desde dentro de la misma expresión regular, utilizando una barra invertida y el número de grupo de captura.

let repeatNum = "42 42 42"; let reRegex = /^(\d+) \1 \1$/; let result = reRegex.test(repeatNum);	Utiliza los grupos de captura en reRegex para que coincida con una cadena que conste sólo del mismo número repetido exactamente tres veces separado por espacios.

32.- 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 el texto con el que coincide. Puedes buscar y reemplazar una cadena de texto usando el método .replace(). Al que le pasaremos como parámetro, en primer lugar la expresión regular y en segundo la cadena para reemplazar la coincidencia.

let str = "one two three"; let fixRegex = /(\w+)\s(\w+)\s(\w+)/; let replaceText = "$3 $2 $1"; let result = str.replace(fixRegex, replaceText);	Escribe una expresión fixRegex utilizando tres grupos de captura que buscarán cada palabra en la cadena one two three. Luego actualiza la variable replaceText para reemplazar one two three por three two one

33.- Elimina espacios en blanco del inicio y final:

let hello = " Hello, World! ";
 	
let wsRegex = /^\s+|\s+$/g;
 	
let result = hello.replace(wsRegex, "");
	Escribe una expresión regular y usa los métodos de cadena apropiados para eliminar espacios en blanco al principio y final de las cadenas. No utilices el método String.prototype.trim().