Regular Expressions(RegExp)

Overview

Regular expressions are the patterns that are used to **match character combinations** in strings. Regular expressions are a powerful way of searching and replace in strings.

In JavaScript, regular expressions are objects. JavaScript provides the built-in RegExp type that allows you to work with regular expressions effectively.

Regular expression allows us to check a string of characters like a password for patterns to see if the set password matches with the pattern defined by that regular expression. Regular expressions are created using **forward slashes (/)** to enclose the pattern.

Creating a regular expression

To create a regular expression in JavaScript, you enclose its pattern in **forward-slash (/)** characters.

```
let reg = /hello/ ;
```

Or using the RegExp constructor:

```
let reg = new RegExp('hello');
```

Modifiers

Modifier	Description
g	Perform a global match (find all matches rather than stopping after the first match)
i	Perform case-insensitive matching
m	Perform multiline matching

Creating a regular expression with modifiers

Example : let reg = /hello/**g** ;
let reg = /hello/**i** ;
let reg = /hello/**m** ;

- /hello/i is a regular expression. “hello” is a pattern, and “i” is a modifier that modifies the search to be case-insensitive. If we write /hello/g, here “g” performs a global match that will find all matches rather than stopping after the first match.

Regular Expressions Methods

Regular expressions are used with the RegExp methods like test( ) and exec( ) and with the string methods replace( ) and split( ) .

Method	Description
exec( )	Tests for a match in a string. Returns the first match
test( )	Tests for a match in a string. Returns true or false
search( )	Returns index of first match else -1
replace( )	Replaces the matched substring with a replacement substring.
split( )	Break a string into an array of substrings

exec( )

This method will execute a search for a match in a string. It returns an **array of information on match or null on a mismatch.**

```
let str = "Welcome to Coding Ninjas" ;
let reg = /Ninjas/ ;
console.log( reg.exec(str) );

Output : ["Ninjas", index: 18, input: "Welcome to Coding Ninjas", groups: undefined]
```

But what if our regular expression was ninjas and not Ninjas

```
let reg = /ninjas/ ;
console.log( reg.exec(str) );
```

Output: null

Now, we can use the **case-insensitive modifier(i)** to search the pattern in the string.

```
let reg = /ninjas/i ;
console.log( reg.exec(str) );
```

Output : ["Ninjas", index: 18, input: "Welcome to Coding Ninjas", groups: undefined]

test( )

The test( ) method tests for a match in a string.
This method returns true if it finds a match; otherwise, it returns false.

```
let str = "Welcome to Coding Ninjas" ;

let reg = /Ninjas/ ;
reg.test(str) ; // Returns true

reg = /hello/ ;
reg.test(str) ; //Returns false
```

search( )

This method returns index of **first match else -1**

```
let str = "Welcome to Coding Ninjas" ;

let reg = /Ninjas/ ;
str.search(reg) ;   // returns 18

reg = /hello/ ;
str.search(reg) ;  // // returns -1
```

replace( )

This method executes a search for a match in a string and replaces the matched substring with a replacement substring.

```
var str = "WELCOME TO CODING NINJAS" ;
var reg = /coding/i ;
str.replace(reg,"Programming"); // WELCOME TO Programming NINJAS
```

split( )

This method uses a regular expression or a fixed string to break a string into an array of substrings.

```
let str = "Welcome to Coding Ninjas"
const reg = /[\s,]+/ ;
let res = str.split(reg) ;
console.log(res);

Output :  [ 'Welcome', 'to', 'Coding', 'Ninjas' ]
```