RegEx Object

- Regular Expressions, commonly known as RegEx, are powerful patterns used to match and manipulate strings in JavaScript and many other programming languages.
- They provide a concise and flexible way to search, validate, and modify text data. In JavaScript, RegEx is supported through the `` `RegExp` `` object and a set of methods for working with patterns.

Creating a RegEx object:

To create a RegEx object in JavaScript, you can use the `` `RegExp` `` constructor or use the literal syntax, which involves enclosing the pattern within forward slashes (/pattern/). For example:

```
// Using the RegExp constructor
var regex = new RegExp("pattern");


// Using the literal syntax
var regex = /pattern/;
```

Common RegEx methods in JavaScript:

- test(): This method checks if a pattern exists within a string and returns a boolean value.

  ```
  var regex = /pattern/;
  var text = "This is a pattern example.";
  var isMatch = regex.test(text);
  console.log(isMatch);  // Output: true
  ```

- exec(): This method searches for a pattern match within a string and returns an array containing the matched result. It also provides additional information about the match, such as the index and input string.

```
var regex = /pattern/;

var text = "This is a pattern example.";

var result = regex.exec(text);

console.log(result);  // Output: ["pattern", index: 10,
input: "This is a pattern example."]
```

- match(): This method searches for a pattern match within a string and returns an array of all matched occurrences.

```
var regex = /pattern/g;

var text = "This is a pattern example. Another pattern
example.";

var matches = text.match(regex);

console.log(matches);  // Output: ["pattern", "pattern"]
```

- search(): It searches for a pattern match within a string and returns the index of the first occurrence. If no match is found, it returns -1.

```
var regex = /pattern/;

var text = "This is a pattern example.";

var index = text.search(regex);

console.log(index);  // Output: 10
```

- replace(): This method searches for a pattern match within a string and replaces it with a specified replacement value.

```
var regex = /pattern/;

var text = "This is a pattern example.";
var modifiedText = text.replace(regex, "new pattern");

console.log(modifiedText);  // Output: "This is a new
pattern example."
```

- split(): This method splits a string into an array of substrings based on a specified pattern match.

```
var regex = /[, ]+/;

var text = "Apple, Banana, Orange";

var fruits = text.split(regex);

console.log(fruits);  // Output: ["Apple", "Banana",
"Orange"]
```

These are just a few of the many methods available for working with RegEx in JavaScript. Regular expressions offer a wide range of pattern-matching capabilities, including character classes, quantifiers, anchors, capturing groups, and more. They are a powerful tool for string manipulation and text processing.

Attribute in RegEx

In regular expressions (RegEx), there are several attributes or flags that can be used to modify the behavior of the pattern matching. These attributes are represented by single characters and are added to the end of the RegEx pattern. Here are some commonly used attributes in JavaScript:

- g (global): This attribute performs a global search, meaning it searches for all occurrences of the pattern in the input string, rather than stopping after the first match.

```
var regex = /pattern/g;
```

- i (ignore case): This attribute performs a case-insensitive search, ignoring the difference between uppercase and lowercase characters.

```
var regex = /pattern/i;
```

- m (multiline): This attribute enables multiline mode, which changes the behavior of the `^` and `$` anchors to match the beginning and end of each line, rather than the entire string.

```
var regex = /^pattern/m;
```

- s (dotAll): This attribute allows the dot (.) metacharacter to match newline characters `(\n)`, which is normally not the default behavior.

```
var regex = /pattern/s;
```

- u (unicode): This attribute enables full Unicode matching, including support for Unicode code point escape sequences `(\u{...}).`

```
var regex = /pattern/u;
```

- y (sticky): This attribute enables sticky mode, which restricts the search to match only at the current position in the input string, indicated by the lastIndex property of the RegEx object.

```
var regex = /pattern/y;
```

These attributes can be used individually or in combination. For example, `/pattern/gi` creates a case-insensitive global search. It's important to note that these attributes affect the behavior of the RegEx pattern and how it interacts with the input string during matching.

- To add multiple attributes, you simply append them together to the end of the RegEx pattern. For example: `/pattern/gim`
- To access or modify the attributes of a RegEx object in JavaScript, you can use the `flags` property. For example:

```
var regex = /pattern/gi;
```

```
console.log(regex.flags);  // Output: "gi"
```

These attributes provide flexibility and control over how RegEx patterns are matched against strings in JavaScript, allowing you to tailor the matching behavior to your specific needs.