Regular Expression (RegEx) in JavaScript

A regular expression (RegEx) is a pattern used to match character combinations in strings. In JavaScript, RegEx can be used for tasks like searching for patterns within strings, replacing parts of strings, and validating input formats.

Syntax for Regular Expression (RegEx)

In JavaScript, regular expressions are written between two slashes (`/pattern/`) or by using the `RegExp` constructor.

Examples:

1. **Literal Syntax**: `/pattern/flags`

   let regex = /hello/;

2**. Constructor Syntax**: `new RegExp("pattern", "flags")`

   let regex = new RegExp("hello");

Common Flags in Regular Expressions

Flags are optional parameters that change how the regular expression is interpreted.

- `g` (global): Search for all occurrences in the string, not just the first one.

- `i` (case-insensitive): Case is ignored when matching.

- `m` (multiline): Allows the start `^` and end `$` anchors to match at the start/end of each line, rather than the entire string.

- `u` (Unicode): Enables full Unicode matching.

- `s` (dot-all): Allows the dot (`.`) to match newline characters.

Basic Regular Expression Patterns

1. Character Sets:

- `[abc]`: Matches any one character from the set `a`, `b`, or `c`.
- `[^abc]`: Matches any character **not** in the set `a`, `b`, or `c`.
- `[0-9]`: Matches any digit from `0` to `9`.
- `[a-z]`: Matches any lowercase letter from `a` to `z`.

2. Special Characters:

- `.`: Matches any character except a newline.
- `\d`: Matches any digit (equivalent to `[0-9]`).
- `\D`: Matches any non-digit character.
- `\w`: Matches any word character (alphanumeric + underscore).
- `\W`: Matches any non-word character.
- `\s`: Matches any whitespace character (spaces, tabs, etc.).
- `\S`: Matches any non-whitespace character.

3. Anchors:

- `^`: Matches the beginning of the string.
- `$`: Matches the end of the string.

4. Quantifiers:

- `*`: Matches 0 or more occurrences of the preceding character or group.
- `+`: Matches 1 or more occurrences of the preceding character or group.
- `?`: Matches 0 or 1 occurrence of the preceding character or group.
- `{n}`: Matches exactly `n` occurrences.
- `{n,}`: Matches `n` or more occurrences.
- `{n,m}`: Matches between `n` and `m` occurrences.

Example Use Cases

1. Finding a Pattern:

```
let str = "Hello, world!";
let regex = /world/;
console.log(regex.test(str)); // true
```

2. Replacing a Pattern:

```
let str = "JavaScript is fun!";
let newStr = str.replace(/fun/, "awesome");
console.log(newStr); // "JavaScript is awesome!"
```

3. Validating Input (e.g., Email):

```
let email = "example@example.com";
let emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
console.log(emailRegex.test(email)); // true
```

Summary on Regular Expressions:

Regular expressions are powerful tools for working with strings in JavaScript, allowing for pattern matching, searching, replacing, and input validation. With various syntax elements like character sets, special characters, anchors, and quantifiers, you can create complex search patterns.