Regular expressions and pattern matching

✅ Regular Expressions and Pattern Matching in JavaScript – *Detailed Explanation*

Regular expressions (commonly known as **RegEx**) are patterns used to match character combinations in strings. In JavaScript, they are a **powerful tool for searching, validating, extracting, and replacing** text.

🔷 1. Creating Regular Expressions

There are two ways to create a RegEx in JavaScript:

```
// Using regex literal
const regex1 = /abc/;

// Using RegExp constructor
const regex2 = new RegExp("abc");
```

🔷 2. Common RegEx Flags

Flag	Description
`g`	Global search (match all occurrences)
`i`	Case-insensitive match
`m`	Multiline match
`s`	Allows `.` to match newline characters
`u`	Unicode support
`y`	"Sticky" search

```
const str = "Hello hello";
const pattern = /hello/gi;
```

🔷 3. Basic Patterns and Metacharacters

Pattern	Matches
`.`	Any single character except newline
`\d`	Any digit (0–9)
`\D`	Non-digit character
`\w`	Word character (a-z, A-Z, 0-9, _)
`\W`	Non-word character
`\s`	Whitespace
`\S`	Non-whitespace
`^`	Start of string
`$`	End of string
`[]`	Character class
`[^]`	Negated character class
`` ` ``	`` ` ``
`()`	Grouping
`{n}`	Exactly n times
`{n,}`	At least n times
`{n,m}`	Between n and m times
`*`	0 or more times
`+`	1 or more times
`?`	0 or 1 time

✅ Examples:

```
const str = "I have 2 cats and 3 dogs.";
```

```
// Find all digits
console.log(str.match(/\d/g)); // ["2", "3"]

// Match words starting with 'c'
console.log(str.match(/\bc\w+/g)); // ["cats"]

// Validate email
const emailPattern = /^[\w.-]+@[a-zA-Z_-]+?\.[a-zA-Z]{2,3}$/;
emailPattern.test("example@gmail.com"); // true
```

🔷 4. String Methods Using RegEx

String.prototype.match()

```
const text = "My age is 25.";
text.match(/\d+/); // ["25"]
```

String.prototype.replace()

```
text.replace(/\d+/, "26"); // "My age is 26."
```

String.prototype.search()

```
text.search(/\d+/); // 11 (position of match)
```

String.prototype.split()

```
"My-name-is-Abhi".split(/-/); // ["My", "name", "is", "Abhi"]
```

🔷 5. RegExp Object Methods

```
const regex = /\d+/g;
regex.test("Age: 25");       // true
regex.exec("Age: 25");       // ["25"]
```

- `test()` → returns `true` or `false`
- `exec()` → returns match info or `null`

🔷 6. Named Capture Groups (ES2018)

```
const date = "2025-06-22";
const pattern = /(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/;
const result = date.match(pattern);

console.log(result.groups.year); // "2025"
```

🔷 7. Practical Use Cases

- **Form validation** (emails, passwords, phone numbers)
- **Search and replace**
- **Data extraction** (e.g., pulling hashtags, mentions)
- **Text formatting**
- **Log or input parsing**

🔷 8. Online Tools

- regex101.com
- RegExr

These tools help you write and test patterns interactively.

🧠 Tips

- Always test RegEx patterns carefully—it's easy to write incorrect or inefficient ones.
- Avoid overly complex RegEx for validation; consider using built-in parsing or libraries where needed.

Would you like sample RegEx patterns for validating email, phone, or password inputs?