

# Regular Expressions (Regex) Basics

---

Regular expressions, often called **regex** or **regexp**, are powerful tools for pattern matching and text manipulation. They are used in various programming languages and tools to search, validate, and manipulate text based on specific patterns.

## Key Concepts

### Using Forward and Backward Slashes

- **Forward Slash (/):** In most programming languages and environments, forward slashes (/) are commonly used as delimiters (character or sequence of characters used to mark the beginning and end of a pattern) to enclose regex patterns.

- **For example:**

`/pattern/`: This defines a regex pattern.

- **Backward Slash (\):** Backward slashes (\) are used to **escape special characters** or to indicate that a character should be treated as a literal character rather than a metacharacter.

- **For example:**

`\\` - Escapes the backslash itself.

`\.` - Matches a literal period (dot).

### Matching Characters Literally

- To match characters literally, you can place them in the regex pattern without special symbols.

- **For example:**

`abc`: Matches the string "abc" exactly as it appears.

### "Or" Operator (|)

- The | (pipe) symbol is an **"or"** operator in regex.

- It allows you to match one of several alternatives.

- **For example:**

cat|dog: Matches either "cat" or "dog."

## "And" Operator (Concatenation)

- In regex, you can use concatenation to create an **"and"** operation between characters or character classes.

- **For example:**

ab: Matches "a" followed by "b."

## Character Classes

- Square brackets [] are used to define character classes.
- Inside a character class, you can list characters or ranges you want to match.

- For example:

[aeiou]: Matches any vowel.

[0-9]: Matches any digit.

[A-Za-z]: Matches any uppercase or lowercase letter.

## Escaping Special Characters

- Special characters in regex (e.g., ., \*, +, ?, (, ), [, ], {, }, \, |) have special meanings.
- To match these characters literally, you need to escape them with a backslash \.

- **For example:**

\. - Matches a literal period.

## Quantifiers

- Quantifiers control the number of occurrences of a character or group.
- Common quantifiers include \* (zero or more), + (one or more), ? (zero or one), {n} (exactly n times), {n,} (at least n times), and {n,m} (between n and m times).

## Anchors

- Anchors are used to specify where a match should occur in the input text.
- `^` matches the start of a line, and `$` matches the end of a line.

## Grouping

- Parentheses `()` are used for grouping and capturing subpatterns.
- They allow you to apply quantifiers and other operations to a group of characters.

## Use Cases:

### Password Matching

The regular expression checks if the input string meets the following criteria for a strong password:

- Contains at least one lowercase letter.
- Contains at least one uppercase letter.
- Contains at least one digit.
- Contains at least one special character from the set `@$!%*?&.`
- It is at least 8 characters long.

```
const mongoose = require("mongoose");

// Define a schema for password strength checking
export const passwordSchema = new mongoose.Schema({
  password: {
    type: String,
    required: true,
    validate: {
      validator: function (value) {
        return
        /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/ .test(value);
      }
    }
  }
});
```

```
    },  
    message: "Password should be between 8-12 characters and  
have a special character",  
  },  
},  
});
```

### Explanation:

The regex pattern

`/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/` enforces strong password criteria.

- **^ and \$:** These anchors match the start and end of the string, ensuring the entire password adheres to the pattern.
- **(?=.\*[a-z]):**
  - `?=` is the syntax for a positive lookahead assertion. It specifies a condition that must be true for the match to continue.
  - `.` matches any sequence of characters (except for line terminators).
  - A positive lookahead assertion checks if the password contains at least one lowercase letter.
- **(?=.\*[A-Z]):** A positive lookahead assertion checks if the password contains at least one uppercase letter.
- **(?=.\*\d):** A positive lookahead assertion checks if the password contains at least one digit.
- **(?=.\*[@\$!%\*?&]):** A positive lookahead assertion checks if the password contains at least one special character from the set `@$!%*?&`.
- **[A-Za-z\d@\$!%\*?&]{8,}:** Matches the password, requiring at least 8 characters from the specified character classes.

### References:

For additional insights on regular expressions, you can refer to the documentation available on [MDN \(Mozilla Developer Network\)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions).