Regular Expressions Cheat Sheet

Regular Expressions Cheat Sheet

Basic Syntax

- `abc` : Matches exactly “abc”
- `.` : Matches any single character except newline
- `^` : Matches the start of the string
- `$` : Matches the end of the string
- `\` : Escapes a special character

Character Classes

- `[abc]` : Matches any one of the characters ‘a’, ‘b’, or ‘c’
- `[^abc]` : Matches any character except ‘a’, ‘b’, or ‘c’
- `[a-z]` : Matches any lowercase letter
- `[A-Z]` : Matches any uppercase letter
- `[0-9]` : Matches any digit
- `[a-zA-Z0-9]` : Matches any alphanumeric character

Predefined Character Classes

- `\d` : Matches any digit (equivalent to `[0-9]`)
- `\D` : Matches any non-digit
- `\w` : Matches any word character (alphanumeric plus underscore, equivalent to `[a-zA-Z0-9_]`)
- `\W` : Matches any non-word character
- `\s` : Matches any whitespace character (space, tab, newline)
- `\S` : Matches any non-whitespace character

Quantifiers

- `*` : Matches 0 or more repetitions of the preceding element

- `+` : Matches 1 or more repetitions of the preceding element
- `?` : Matches 0 or 1 repetition of the preceding element
- `{n}` : Matches exactly n repetitions
- `{n,}` : Matches n or more repetitions
- `{n,m}` : Matches between n and m repetitions

Groups and Alternation

- `(abc)` : Matches the string “abc” and remembers the match
- `a|b` : Matches either ‘a’ or ‘b’
- `(?:abc)` : Matches “abc” but does not remember the match (non-capturing group)
- `(?=abc)` : Positive lookahead; matches “abc” but does not consume it
- `(?!abc)` : Negative lookahead; ensures that the string is not “abc” but does not consume it

Common Patterns

1. **Email Validation**:

```
/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/
```

   - Matches a typical email address format.

2. **Phone Number Validation (10 digits)**:

```
/^\d{10}$/
```

   - Matches exactly 10 digits.

3. **Password Validation**:

```
/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{8,}$/
```

   - Ensures the password contains at least one lowercase letter, one uppercase letter, one digit, and is at least 8 characters long.

4. **URL Validation**:

```
/^(https?:\/\/)?([\da-z.-]+)\.([a-z.]{2,6})([\/\w .-]*)*\/?$/
```

- Matches a typical URL format.

5. **Date Validation (YYYY-MM-DD)**:

```
/^\d{4}-\d{2}-\d{2}$/
```

- Matches dates in the format YYYY-MM-DD.

6. **Time Validation (HH:MM)**:

```
/^([01]\d|2[0-3]):([0-5]\d)$/
```

- Matches time in the format HH:MM (24-hour format).

Special Characters

- `\b` : Matches a word boundary
- `\B` : Matches a non-word boundary
- `\n` : Matches a newline character
- `\t` : Matches a tab character
- `\0` : Matches a null character

Examples of Use in JavaScript

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

// Example: Validate Password
const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]
{8,}$/;
```

```
const password = "Password123";
console.log(passwordRegex.test(password)); // true
```

Tips

- Use tools like Regex101 to test and debug your regex patterns.
- Remember that regex can be complex; start simple and build up your patterns as needed.
- Comments can be added to complex regex patterns using the `(?x)` flag, but this is not supported in all regex engines.

This cheat sheet should help you get started with regular expressions and serve as a quick reference for your future projects.