



## Regex Fundamentals

### Basic Patterns

<code>abc</code>	Matches the literal sequence <code>abc</code> .
<code>[abc]</code>	Matches any single character: <code>a</code> , <code>b</code> , or <code>c</code> .
<code>[^abc]</code>	Matches any single character <i>except</i> <code>a</code> , <code>b</code> , or <code>c</code> .
<code>[a-z]</code>	Matches any lowercase letter from <code>a</code> to <code>z</code> .
<code>[0-9]</code>	Matches any digit from <code>0</code> to <code>9</code> .
<code>.</code>	Matches any single character (except newline).

### Metacharacters

<code>\d</code>	Matches any digit (same as <code>[0-9]</code> ).
<code>\D</code>	Matches any non-digit character (same as <code>[^0-9]</code> ).
<code>\w</code>	Matches any word character (alphanumeric and underscore, same as <code>[a-zA-Z0-9_]</code> ).
<code>\W</code>	Matches any non-word character (same as <code>[^a-zA-Z0-9_]</code> ).
<code>\s</code>	Matches any whitespace character (space, tab, newline).
<code>\S</code>	Matches any non-whitespace character.

### Anchors

<code>^</code>	Matches the beginning of the string.
<code>\$</code>	Matches the end of the string.
<code>\b</code>	Matches a word boundary (the position between a word character and a non-word character).
<code>\B</code>	Matches a non-word boundary.

## Quantifiers and Grouping

### Quantifiers

<code>*</code>	Matches the preceding element 0 or more times.
<code>+</code>	Matches the preceding element 1 or more times.
<code>?</code>	Matches the preceding element 0 or 1 time.
<code>{n}</code>	Matches the preceding element exactly <code>n</code> times.
<code>{n,}</code>	Matches the preceding element <code>n</code> or more times.
<code>{n,m}</code>	Matches the preceding element between <code>n</code> and <code>m</code> times (inclusive).

### Grouping and Capturing

<code>()</code>	Groups the enclosed pattern. Captures the matched text for backreferencing.
<code>(?:pattern)</code>	Non-capturing group. Groups the pattern without capturing the matched text.
<code> </code>	Acts as an 'or' operator. Matches either the pattern before or after the <code> </code> .
<code>(?&lt;name&gt;...)</code>	Named capturing group. Matches <code>...</code> and stores it in the group named <code>name</code> .
<code>\1, \2, ...</code>	Backreferences to the captured groups. <code>\1</code> refers to the first captured group, <code>\2</code> to the second, and so on.

### Greedy vs. Lazy Matching

By default, quantifiers are <i>greedy</i> , meaning they match as much as possible.
Add a <code>?</code> after a quantifier to make it <i>lazy</i> , matching as little as possible.
<b>Example:</b>
Given the string <code>&lt;a&gt;&lt;b&gt;&lt;/a&gt;&lt;/b&gt;</code> and the pattern <code>&lt;.*&gt;</code> :
<ul style="list-style-type: none"><li>Greedy: matches <code>&lt;a&gt;&lt;b&gt;&lt;/a&gt;&lt;/b&gt;</code></li><li>Lazy: matches <code>&lt;a&gt;</code></li></ul>

## Advanced Regex Features

### Lookarounds

<code>(? =pat tern)</code>	Positive lookahead assertion. Ensures that the pattern is followed by <code>pattern</code> , but doesn't include <code>pattern</code> in the match.
<code>(? !pattern)</code>	Negative lookahead assertion. Ensures that the pattern is <i>not</i> followed by <code>pattern</code> .
<code>(? &lt;=pattern)</code>	Positive lookbehind assertion. Ensures that the pattern is preceded by <code>pattern</code> , but doesn't include <code>pattern</code> in the match (not supported in all regex engines).
<code>(? &lt;!pattern)</code>	Negative lookbehind assertion. Ensures that the pattern is <i>not</i> preceded by <code>pattern</code> (not supported in all regex engines).

### Flags/Modifiers

<code>i</code>	Case-insensitive matching.
<code>g</code>	Global matching (find all matches, not just the first).
<code>m</code>	Multiline matching. <code>^</code> and <code>\$</code> match the start and end of each line (as well as the start/end of the string).
<code>s</code>	Dotall. Allows <code>.</code> to match newline characters.

### Conditional Regex

<code>(?(condition)then else)</code>	- Matches the <code>then</code> part if the <code>condition</code> is met, otherwise matches the <code>else</code> part. The <code>else</code> part can be omitted.
--------------------------------------	---

## Common Regex Operations

### Substitution

Replace matches of a pattern with a specified string.

Example (Python):

```
import re

text = "The quick brown fox"
new_text = re.sub(r"\s+", "-", text)
print(new_text) # Output: The-quick-
                brown-fox
```

### Splitting

Split a string into a list of substrings based on a regex delimiter.

Example (JavaScript):

```
const text = "apple,banana,orange";
const fruits = text.split(/,/);
console.log(fruits); // Output: [
  'apple', 'banana', 'orange' ]
```

### Validation

Verify that a string matches a specific format using regex.

Example (Java):

```
import java.util.regex.Pattern;

String email = "test@example.com";
boolean isValid = Pattern.matches("[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}", email);
System.out.println(isValid); // Output:
true
```