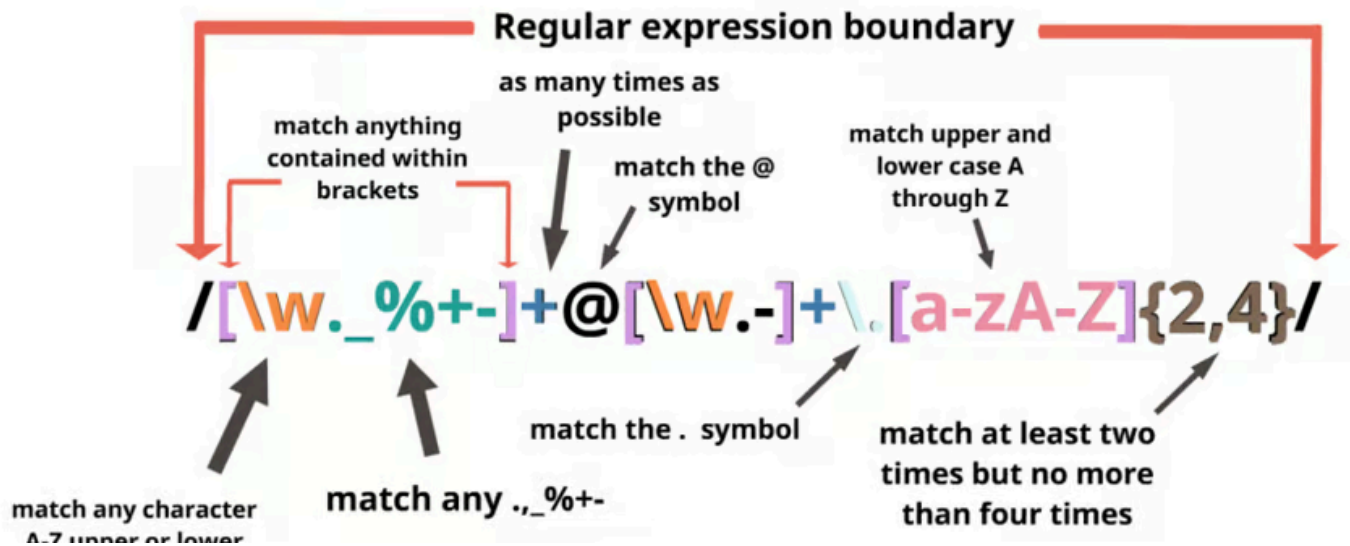




## 1.4.2 Regex Tutorial



Regular expressions (regex) are a powerful tool for pattern matching and text processing.

### 1. Basic Syntax

- **Literal Characters:** The simplest regex is just a string of literal characters. For example, "cat" matches the string "cat".
- **Metacharacters:** Characters that have special meaning in regex:
  - `.`: Matches any single character except a newline.
  - `^`: Matches the start of a string.
  - `$`: Matches the end of a string.
  - `\`: Escape character to treat special characters as literals (e.g., `\.` to match a period).

### 2. Character Classes

- **Square Brackets []**: Match any one character inside the brackets.
  - `[abc]`: Matches "a", "b", or "c".
  - `[a-z]`: Matches any lowercase letter from "a" to "z".
  - `[^abc]`: Negated class; matches any character except "a", "b", or "c".

### 3. Quantifiers

Quantifiers specify how many times a character or group must be present.

- `*`: 0 or more times (e.g., `a*` matches "", "a", "aa", "aaa").
- `+`: 1 or more times (e.g., `a+` matches "a", "aa", "aaa").

- `?`: 0 or 1 time (e.g., `a?` matches "" or "a").
- `{n}`: Exactly `n` times (e.g., `a{3}` matches "aaa").
- `{n,}`: `n` or more times (e.g., `a{2,}` matches "aa", "aaa", "aaaa").
- `{n,m}`: Between `n` and `m` times (e.g., `a{2,4}` matches "aa", "aaa", or "aaaa").

## 4. Anchors

- `^`: Asserts the position at the start of a line (e.g., `^cat` matches "cat" only at the start).
- `$`: Asserts the position at the end of a line (e.g., `cat$` matches "cat" only at the end).

## 5. Grouping and Alternation

- **Parentheses** `()`: Used for grouping and capturing.
  - `(abc)` captures "abc".
  - `(a|b|c)` matches "a", "b", or "c" (alternation).
- **Non-capturing group** `(?:...)`: Groups without capturing.

## 6. Common Functions in R

R has several functions for working with regex, primarily in the `stringr` and `base` packages:

### `grep`: Search for patterns in strings

```
text <- c("cat", "bat", "rat")
grep("b", text) # Returns indices of matches
```

```
[1] 2
```

### `grep1`: Logical vector indicating matches

```
grep1("c", text) # Returns TRUE or FALSE
```

```
[1] TRUE FALSE FALSE
```

### `gsub` / `sub`: Replace matches with a new string

```
gsub("c", "C", text) # Replaces "c" with "C" in all elements
```

```
[1] "Cat" "bat" "rat"
```

### `regexpr` / `gregexpr`: Returns the position and length of matches

```
regexpr("at", text) # Returns the starting position and length of the match
```

```
[1] 2 2 2
attr(,"match.length")
```

```
[1] 2 2 2
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE
```

## `str_extract` (from `stringr`): Extracts matching substrings

```
library(stringr)
str_extract("Price: $123.45", "\\$[0-9]+\\. [0-9]{2}") # Extracts "$123.45"
```

```
[1] "$123.45"
```

## 7. Examples

- **Matching an email address:**

```
emails <- c("user@example.com", "another.user@domain.co")
email_pattern <- "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}"
grep(email_pattern, emails)
```

```
[1] 1 2
```

**emails**: This is a character vector that contains two strings, each representing an email address.

- "user@example.com"
- "another.user@domain.co"

Let's break down the components of the regex pattern:

`[a-zA-Z0-9._%+-]+`:

- `[a-zA-Z0-9._%+-]`: This character class matches any of the following characters:
  - `a-z`: Any lowercase letter.
  - `A-Z`: Any uppercase letter.
  - `0-9`: Any digit.
  - `.`: A literal period (dot).
  - `_`: An underscore.
  - `%`: A percentage sign.
  - `+`: A plus sign.
  - `-`: A hyphen.

- **+**: This quantifier means "one or more" of the preceding characters.

This part of the pattern matches the local part of the email address (the part before the @ symbol).

**@**:

- This matches the literal @ symbol that separates the local part of the email from the domain part.

**[a-zA-Z0-9.-]+**:

- **[a-zA-Z0-9.-]**: This character class matches any of the following:
  - **a-z**: Any lowercase letter.
  - **A-Z**: Any uppercase letter.
  - **0-9**: Any digit.
  - **.**: A literal period (dot).
  - **-**: A hyphen.
- **+**: This quantifier means "one or more" of the preceding characters.

This part of the pattern matches the domain name part of the email address.

**\\.:**

- **\\.:** Matches a literal period (dot). The double backslash is used to escape the period in R, as the period is a special character in regex.

**[a-zA-Z]{2,}**:

- **[a-zA-Z]**: Matches any letter, either lowercase (**a-z**) or uppercase (**A-Z**).
- **{2,}**: This quantifier means "two or more" of the preceding character class.

This part of the pattern matches the top-level domain (TLD) part of the email address, which typically consists of two or more letters (like ".com" or ".co").

- **Extracting phone numbers:**

```
text <- "Call me at 555-123-4567 or 555.987.6543"
phone_pattern <- "\\b[0-9]{3}[-.]?[0-9]{3}[-.]?[0-9]{4}\\b"
str_extract_all(text, phone_pattern)
```

```
[[1]]
```

```
[1] "555-123-4567" "555.987.6543"
```

Let's break down the components of this pattern:

`\\b`:

- `\\b`: This is a word boundary anchor. It matches the position where a word starts or ends. This ensures that the phone number pattern is matched as a complete word, not as a part of a longer string of digits.

`[0-9]{3}`:

- `[0-9]`: This matches any digit from 0 to 9.
- `{3}`: This quantifier specifies that exactly three digits should be matched.

This part of the pattern matches the first three digits of the phone number.

`[-.]?`:

- `[-.]`: This matches either a hyphen ( - ) or a period ( . ).
- `?`: This quantifier specifies that the hyphen or period is optional (it can appear 0 or 1 time).

This part of the pattern matches the separator between the first three digits and the next three digits of the phone number. It allows for flexibility in formatting (e.g., "555-123-4567" or "555.987.6543").

`[0-9]{3}`:

- Same as before, this matches the next three digits of the phone number.

`[-.]?`:

- Again, this matches an optional hyphen or period separator.

`[0-9]{4}`:

- This matches the final four digits of the phone number.

`\\b`:

- The word boundary anchor ensures that the pattern matches the end of a complete phone number.

## 8. Escape Characters

- The slashes ( `\` ) is used to escape certain characters in the regular expression so that they are treated as literal characters rather than as special regex symbols, for example the dollar sign \$ is a special character in regex, typically used as an anchor to denote the end of a line or a string, therefore we escape it with a backslash if we want to capture the \$ character in a string.
- The backslash itself is also a special character, so you need to escape the backslash. This means you write `\\$` to match a literal \$ in the string.
- So, `\\$` matches the literal dollar sign ( \$ ) in the string.

## 9. Advanced Topics

- **Lookahead and Lookbehind:** Zero-width assertions that allow you to match something only if it's preceded or followed by another pattern.
  - Positive Lookahead: `(?=...)`
  - Negative Lookahead: `(?!...)`
  - Positive Lookbehind: `(?<=...)`
  - Negative Lookbehind: `(?<!...)`
- **Lazy Quantifiers:** By default, quantifiers are greedy (they match as much as possible). Add `?` after a quantifier to make it lazy (e.g., `. * ?`).

## Summary

Regex in R is a versatile tool for text processing. The basics include character classes, quantifiers, and grouping. R provides several functions to work with regex effectively. With practice, you can craft complex patterns for searching, extracting, and manipulating text data.