

# The general idea

A **regular expression** is a pattern that a string may or may not match.

Example: [0-9] +

[0-9] means a character in that range

“+” means one or more of what came before

Strings that match: 9125 4

Strings that don't: abc empty string

Symbols like [, ], and + have special meaning. They are not part of the string that matches. (If we want them to be, we “escape” them with a backslash.)

# Example Uses

## Handling white space

A program ought to be able to treat any number of white space characters as a separator.

## Identifying blank lines

Most people consider a line with just spaces on it to be blank.

## Validating input

To check that input is has the expected format (e.g., a date in DD-MM-YYYY format).

## Finding something within some input

E.g., finding dates within paragraphs of text.

# Why bother?

We could accomplish those tasks without using patterns.

But it's much easier to declare a pattern that you want matched than to write code that matches it.

Therefore many languages offer support for this.

Bonus: By having the pattern explicitly declared, rather than implicit in code that matches it, it's much easier to:

understand what the pattern is

modify it

# Editors, unix, Python, Java...

Regular expressions are used in many places.

Editors like vi, emacs, and Sublime Text allow you to use regular expressions for searching.

Many unix commands use regular expressions. Example:

`grep pattern file`

prints all lines from file that match pattern

Many programming languages provide a library for regular expressions.

The syntax varies from context to context, but the core is the same everywhere.

# Simple patterns

Pattern	Matches	Explanation
a*	“ ‘a’ ‘aa’	zero or more
b+	‘b’ ‘bb’	one or more
ab?c	‘ac’ ‘abc’	zero or one
[abc]	‘a’ ‘b’ ‘c’	one from a set
[a-c]	‘a’ ‘b’ ‘c’	one from a range
[abc]*	“ ‘acbccb’	combination

Note: In Java, patterns can be used to match an occurrence anywhere in the string, or one that consumes the whole string, among other options.

# Anchoring

Lets you force the position of match

$\wedge$  matches the beginning of the line

$\$$  matches the end

Neither consumes any characters.

<b>Pattern</b>	<b>Text</b>	<b>Result</b>
b+	abbc	Matches
$\wedge b+$	abbc	Fails (no b at start)
$\wedge a^* \$$	aabaa	Fails (not all a's)

# Escaping

Match actual

^ and \$ and [ etc.

using escape sequences

\^ and \\$ and \[ etc.

Remember, we also use escapes for other characters:

\t is a tab character

\n is a newline

# Predefined Character Classes

Construct	Description
.	any character
\d	a digit [0-9]
\D	a non-digit [^0-9]
\s	a whitespace char [ \t\n\x0B\f\r]
\S	a non-whitespace char [^\s]
\w	a word char [a-zA-Z_0-9]
\W	a non-word char [^\w]

The notation [ ^abc ] means “anything not in the set”.

# Defining your own Character Classes

Construct	Description
[abc]	a, b, or c (simple class)
[^abc]	any char except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z inclusive (range)
[a-d[m-p]]	a through d or m through p (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-zA-Z&&[^bc]]	a through z except for b and c (subtraction)
[a-zA-Z&&[^m-p]]	a through z and not m through p (subtraction)

# Quantifiers

Construct	Description
$X?$	0 or 1 times
$X^*$	0 or more times
$X^+$	1 or more times
$X\{n\}$	exactly n times
$X\{n,\}$	at least n times
$X\{n,m\}$	at least n but no more than m times

# Capturing Groups and Backreferences

**Capturing groups** allow you to treat multiple characters as a single unit.

Use **parentheses** to group.

Capturing groups are **numbered** by counting their opening parentheses from left to right.

`((A)(B(C)))` has the following groups:

1.`((A)(B(C)))`

2.`(A)`

3.`(B(C))`

4.`(C)`

# Capturing Groups and Backreferences

The section of the input string matching the capturing group(s) is saved in memory for later recall via **backreference**.

A backreference is specified in the regular expression as a backslash (\) followed by a digit indicating the number of the group to be recalled.

For example,

Pattern	Example matching string
(\d\d)\ \	I2I2
(\w*)\s\ \	asdf asdf

# Regular expressions in Java

The `java.util.regex` package contains:

`Pattern`: a compiled regular expression

`Matcher`: the result of a match

Example: `RegexDemo`