/r/e/g/e/x

AP Fall-1404 Dr. Mojtaba Vahidi Asl

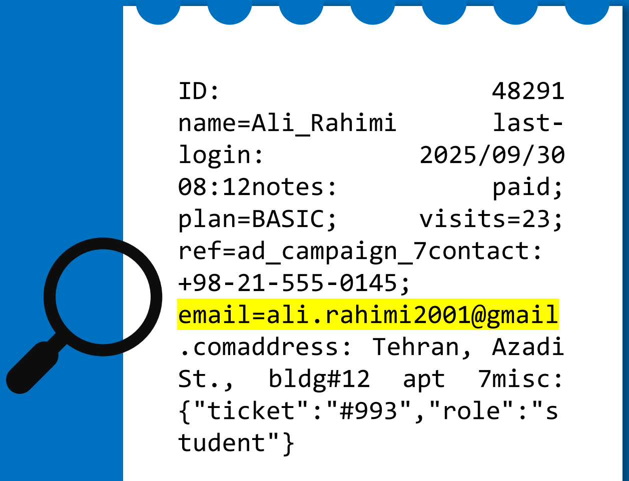
by Danial Yaghooti

Welcome to the World of Regular Expressions (Regex)

Have you ever wanted to find specific words, emails, or numbers inside thousands of lines of text — instantly?

Rege is a powerful tool that lets us describe text patterns and search, extract, or validate them automatically.

It’s used everywhere: from Java programs to ID validation, log analysis, and data cleaning.



```
ID:                  48291
name=Ali_Rahimi      last-
login:          2025/09/30
08:12notes:          paid;
plan=BASIC;      visits=23;
ref=ad_campaign_7contact:
+98-21-555-0145;
email=ali.rahimi2001@gmail
.comaddress: Tehran, Azadi
St.,  bldg#12  apt  7misc:
{"ticket":"#993","role":"s
tudent"}
```

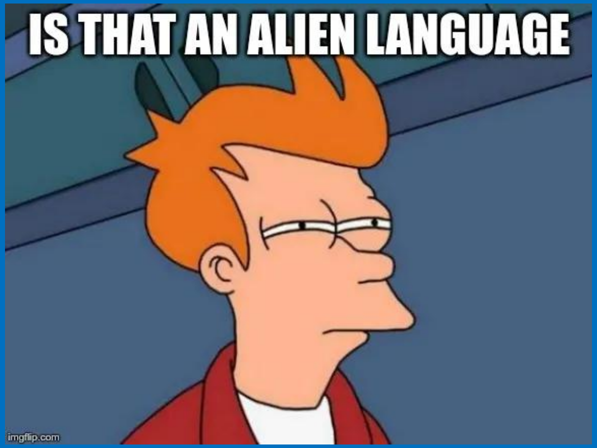
How Regex Works\?

Regex has two main parts:

- Pattern: the rule or formula we design.
- Engine: the part inside Java, Python, … that tests our pattern against text.

What we will learn in this session:

👉 How to build patterns that the engine can understand.



Text

"User 125 Ali@gmail.com 45$"

Pattern

`[\w.+-]+@[\w.-]+\.[A-Za-z]{2,6}`

Regex Engine

Email

Literal Matches: Start Simple

Every pattern starts with the simplest idea — literal characters.

The pattern "aaa" will only match the exact text aaa.

It’s case-sensitive by default. Regex is case-sensitive by default.

Regex:

```
cat
```

Matches:

cat , catalog, concatenate

Flexible Matching with Character Sets

Sometimes, we don’t want to match a fixed letter, but one of several

- `[abc]` → matches one of these characters: a, b, or c.
- `[0-9]` → any digit.
- `[^a-z]` → any character except lowercase letters.
- Regex is about describing “possibilities”.

Regex:

```
gr[ea]y
```

Matches:

gray, grey

Escape Characters

They allow you to match special characters that have a specific meaning in regex.

- `\w` matches any word character [a-z A-Z 0-9 _]
- `\s` matches any whitespace character (equivalent to [\r\n\t\f\v])
- `\d` matches any digits [1-9]

Regex:

```
\d\d-\d\d
```

Matches:

24-50, 1-35, 85-85, 5

How Many Times? — Quantifiers

Quantifiers tell Regex how many times something can repeat.

- `{n}` exactly one time `{n , m}` between n and m times
- `+` one or more times
- `*` Zero or more
- `?` Zero or one

Regex:

```
c?a{2}t*
```

Matches:

cat , caatalog, conaatttenate

Anchors: Position Matters

Regex can also match positions, not just characters.

- `^` start of line
- `$` end of line

So if you want the exact matches you can write `^otherRegex$`

Regex:

```
^[Hh]ey
```

Matches:

```
Wow, hey gorgeous!
Hey you, out there in the cold..
```

Dot, Escapes

- `.` matches *any* character (except newline).
- `\.` Matches literal dot

Regex:

```
\w+\.txt
```

Matches:

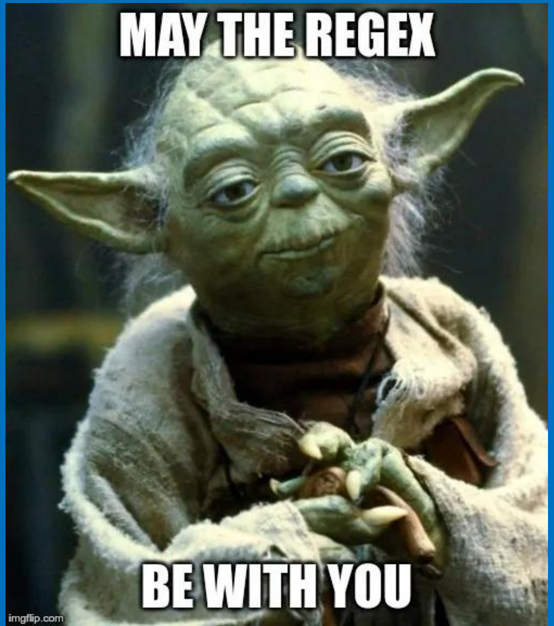
File1.txt , out.json , users.txt , cow.jpg

Grouping and Capturing

- Parentheses `()` let us treat parts of a pattern as one unit.
- `(abc)` repeats “abc” together.
- You can also capture what’s `()` matched using Java’s `Matcher.group()`

Wait What ?!

+ regex



Regex in Java

The two core Classes:

- *Pattern* : Represents a compiled regex pattern.

```
Pattern pattern = Pattern.compile("regex");
```

- *Matcher* : Applies the pattern to a given text.

```
Matcher matcher = pattern.matcher(text);
matcher.find();                  // Find next
matchmatcher.group();            // Get matched
substringmatcher.matches();      // Check full-string
matchmatcher.replaceAll("x");    // Replace all matches
```

Find more details from here

Quick Example

Consider we want toy find the date such as yyyy-mm-dd:

Regex:

```
(\d{4})-(\d{2})-(\d{1,2})
```

Matches:

```
String data= “Emma Stone was born 1988-11-6.”;
String Regex= “(\\d{4})-(\\d{2})-(\\d{1,2})”;

Pattern pattern = Pattern.compile(Regex);
Matcher matcher = pattern.matcher(data);
if(matcher.find())
{
     int year= Integer.parseInt(matcher.group(1));
     int month= Integer.parseInt(matcher.group(2));
     int day= Integer.parseInt(matcher.group(3));
     System.out.println(String.format("Year: %d month: %d day: %d", year, month, day));
}
else System.out.println("No data found!");
```

Non-Capturing Groups

Parentheses ( ) group parts of a regex.

By default, they **capture** the matched text - stored in a group index.

Sometimes, you want grouping **without capturing** so use (?: …)

Regex:

```
(?:\d+)-(\w+)
```

Matches:

```
\\in Java
String data= “14-code”;
Matcher.group(1) = “code” not 14
```

Alternation , Choosing Between Patterns

- `|` pipe means **OR** between expressions.
- `cat|dog` matches either “cat” or “dog”.

Regex:

```
I love (Spiderman|Batman)
```

Matches:

```
I love Spiderman
I love Batman
```

Word Boundary

- `\b` matches a word boundary — the position between a word character (\w) and a non-word character (\W).

Regex:

```
\bcat\b
```

Matches:

cat , catalog, catwoman , dog or cat.

Lookaround Assertions

Lookarounds check context without including it in the match.

- `(?=…)` Lookahead
- `(?! …)` Negative lookahead
- `(?<= …)` Lookbehind
- `(?<! …)` Negative lookbehind

Regex:

```
(?<!\d)\d{4}(?!\d)
```

Matches:

14558 days after the 1988 become 1990.

Where should I test my regex\?

RegExr

Regex101

Thank you for your time [__])