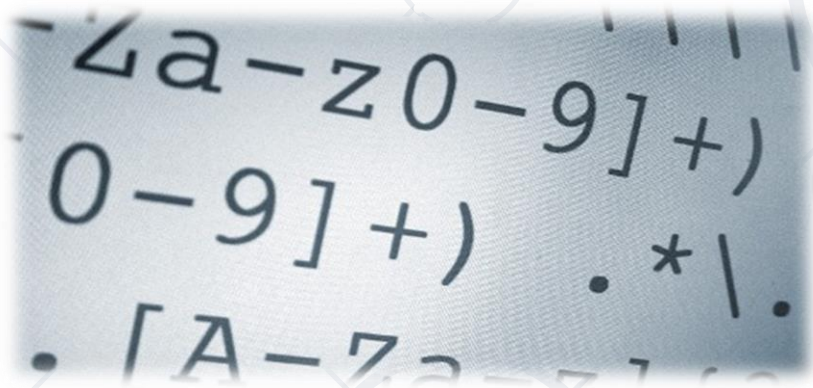


# Regular Expressions (RegEx)

## Regular Expressions Language Syntax



-[a-z0-9]+  
0-9]+)  
.\*\.  
[A-Z0-9\_+@%&#~\.\-:;]{1,64}

SoftUni Team

Technical Trainers



SoftUni



Software University

<https://softuni.bg>

sli.do

**#fund-java**

1. Regular Expressions Syntax
  - Definition and Pattern
  - Predefined Character Classes
2. Quantifiers and Grouping
3. Backreferences
4. Regular Expressions in Java





**[A-Z]**

# **Regular Expressions**

Definition and Classes

# What Are Regular Expressions?

- Regular expressions (regex)
  - Match text by pattern
- Patterns are defined by special syntax, e.g.
  - `[0-9]+` matches non-empty sequence of digits
  - `[A-Z][a-z]*` matches a capital + small letters
- Play with regex live at: [regexr.com](https://regexr.com), [regex101.com](https://regex101.com)



regular expressions 101 @regex101 donate contact bug reports & feedback wiki

REGULAR EXPRESSION 17 matches, 1035 steps (~2ms)

/ [A-Z]\w+ /g

TEST STRING SWITCH TO UNIT TESTS

Edit the Expression & Text to see matches. Roll over matches or the expression for details. Undo mistakes with ctrl-z. Save Favorites & Share expressions with friends or the Community. Explore your results with Tools. A full Reference & Help is available in the Library, or watch the video Tutorial.

Sample text for testing:

abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ

0123456789 \_+-. ,!@#\$%^&\*();\|/|<>"'

12345 -98.7 3.141 .6180 9,000 +42

555.123.4567 +1-(800)-555-2468

# Live Demo

Www.regex101.com

# Regular Expression Pattern – Example

- Regular expressions (regex) describe a search pattern
- Used to find / extract / replace / split data from text by pattern

`[A-Z][a-z]+ [A-Z][a-z]+`

John Smith

Linda Davis

Contact: Alex Scott

# Character Classes: Ranges

- **[nvj]** matches any character that is either **n**, **v** or **j**

**node.js v0.12.2**

- **[^abc]** - matches any character that is **not** **a**, **b** or **c**

**Abraham**

- **[0-9]** - character range matches any digit from **0** to **9**

**John is 8 years old.**



- **\w** - matches any **word character** (a-z, A-Z, 0-9, \_)
- **\W** - matches any **non-word character** (the opposite of \w)
- **\s** - matches any **white-space** character
- **\S** - matches any **non-white-space** character (opposite of \s)
- **\d** - matches any **decimal digit** (0-9)
- **\D** - matches any **non-decimal character** (the opposite of \d)



**(\w+)**

**Quantifiers**

Grouping

- **\*** - matches the previous element zero or more times

`\+\d*` → `+359885976002 a+b`

- **+** - matches the previous element one or more times

`\+\d+` → `+359885976002 a+b`

- **?** - matches the previous element zero or one time

`\+\d?` → `+359885976002 a+b`

- **{3}** - matches the previous element exactly 3 times

`\+\d{3}` → `+359885976002 a+b`

- **(subexpression)** - captures the matched subexpression as numbered group

`\d{2}-(\w{3})-\d{4}` → `22-Jan-2015`

- **(?:subexpression)** - defines a non-capturing group

`^(?:Hi|hello),\s*(\w+)$` → `Hi, Peter`

- **(?<name>subexpression)** - defines a named capturing group

`(?<day>\d{2})-(?<month>\w{3})-(?<year>\d{4})` → `22-Jan-2015`

# Problem: Match All Words

- Write a regular expression in [www.regex101.com](https://www.regex101.com) that extracts all word char sequences from given text

**\_ (Underscores) are  
also word characters!**



**\_|Underscores|are|also|  
word|characters**

# Problem: Match Dates

- Write a regular expression that extracts **dates** from text
  - Valid date format: **dd-MMM-yyyy**
  - Examples: **12-Jun-1999**, **3-Nov-1999**

I am born on **30-Dec-1994**.  
My father is born on the **9-Jul-1955**.  
**01-July-2000** is not a valid date.

# Problem: Email Validation

- Write a regular expression that performs simple **email validation**
  - An email consists of **username @ domain name**
  - **Usernames** are **alphanumeric**
  - **Domain names** consist of **two strings**, separated by a **period**
  - **Domain names** may contain only **English letters**

Valid:

`valid123@email.bg`

Invalid:

`invalid*name@email1.bg`



# Backreferences

Numbered Capturing Group



# Backreferences Match Previous Groups

- **\number** - matches the value of a numbered capture group

```
<(\w+)[^>]*>.*?<\1>
```

```
<b>Regular Expressions</b> are cool!
```

```
<p>I am a paragraph</p> ... some text after
```

```
Hello, <div>I am a<code>DIV</code></div>!
```

```
<span>Hello, I am Span</span>
```

```
<a href="https://softuni.bg/">SoftUni</a>
```



# Regular Expressions

Using Built-In Regex Classes

- Regex in Java library
  - `java.util.regex.Pattern`
  - `java.util.regex.Matcher`

```
Pattern pattern = Pattern.compile("a*b");  
Matcher matcher = pattern.matcher("aaaab");
```

```
boolean match = matcher.find();  
String matchText = matcher.group();
```

Searches for the  
next match

Gets the matched text

# Checking for a Single Match

- **find()** - gets the first pattern match

```
String text = "Andy: 123";  
String pattern = "([A-Z][a-z]+): (?<number>\\d+)";
```

```
Pattern regex = Pattern.compile(pattern);  
Matcher matcher = regex.matcher(text);
```

```
System.out.println(matcher.find());           // true  
System.out.println(matcher.group());          // Andy: 123  
System.out.println(matcher.group(0));        // Andy: 123  
System.out.println(matcher.group(1));        // Andy  
System.out.println(matcher.group(2));        // 123  
System.out.println(matcher.group("number")); // 123
```

**+** - Matches the element one or more times

- To replace **every/first** subsequence of the input sequence that matches the pattern with the given replacement string
  - **replaceAll(String replacement)**
  - **replaceFirst(String replacement)**

```
String regex = "[A-Za-z]+";  
String string = "Hello Java";  
Pattern pattern = Pattern.compile(regex);  
Matcher matcher = pattern.matcher(string);  
String res = matcher.replaceAll("hi");    // hi hi  
String res2 = matcher.replaceFirst("hi"); // hi Java
```

- `split(String pattern)` - splits the text by the pattern
  - Returns `String[]`

```
String text = "1 2 3 4";
```

```
String pattern = "\\s+";
```

Matches whitespaces

```
String[] tokens = text.split(pattern);
```

`tokens = {"1", "2", "3", "4"}`

# Problem: Match Full Name

- You are given a list of names
  - Match all full names

Ivan Ivanov, Ivan ivanov, ivan Ivanov, IVan Ivanov, Georgi Georgiev, Ivan Ivanov



Ivan Ivanov Georgi Georgiev

Check your solution here: <https://judge.softuni.org/Contests/1672/>

# Solution: Match Full Names

```
String listOfNames = reader.readLine();

String regex = "\\b[A-Z][a-z]+ [A-Z][a-z]+";
Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(listOfNames);

while (matcher.find()) {
    System.out.print(matcher.group() + " ");
}
```

Check your solution here: <https://judge.softuni.org/Contests/1672/>



# Problem: Match Dates

- You are given a string
  - Match all dates in the format "**dd{separator}MMM{separator}yyyy**" and print them space-separated

13/Jul/1928, 01/Jan-1951



Day: 13, Month: Jul, Year: 1928

Check your solution here: <https://judge.softuni.org/Contests/1672/>

# Solution: Match Dates

```
String input = reader.readLine();

String regex =
"\b(?:<day>\d{2})(?:\.|\/|-)(?:<month>[A-Z][a-z]{2})\d{2}(?:<year>\d{4})\b";

Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(dates);

while (matcher.find()) {
    System.out.println(String.format("Day: %s, Month: %s, Year: %s",
        matcher.group("day"), matcher.group("month"),
        matcher.group("year")));
}
```

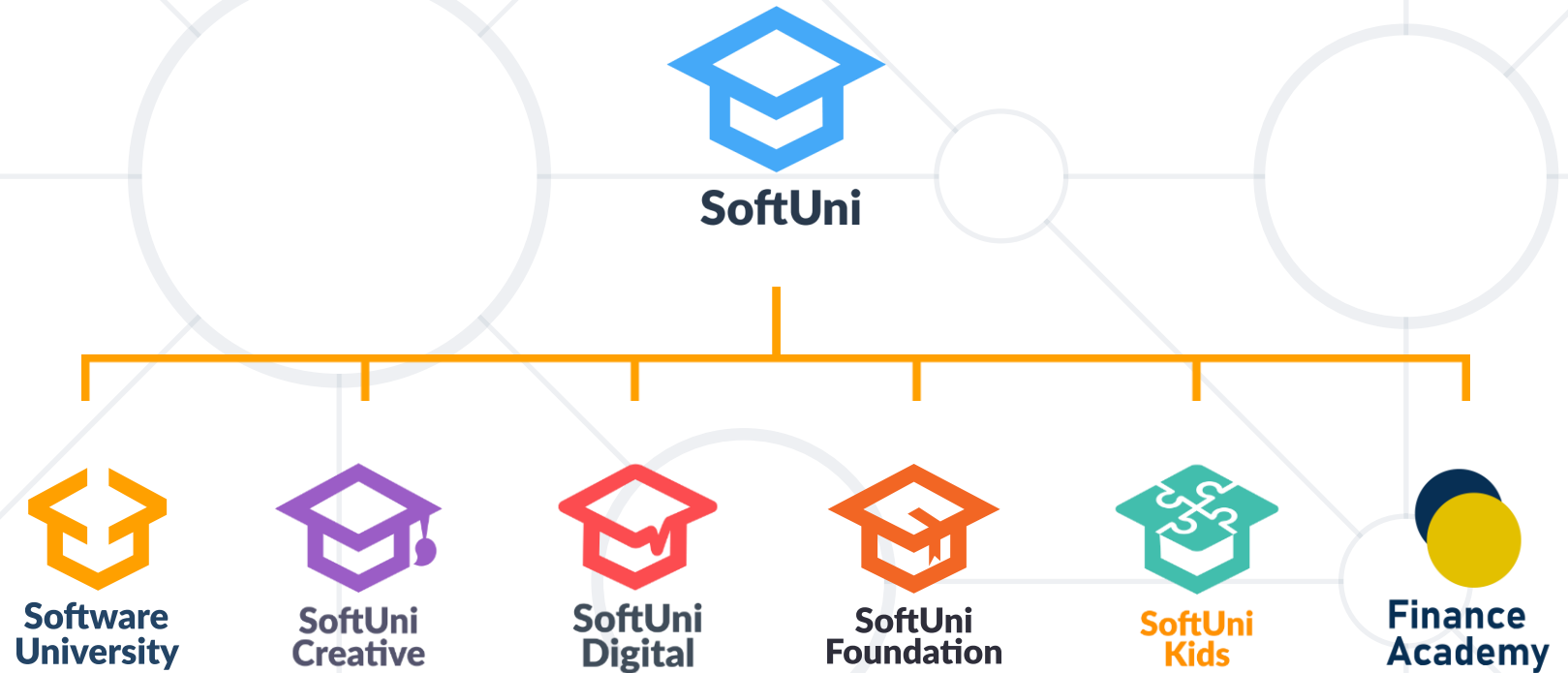
Check your solution here: <https://judge.softuni.org/Contests/1672/>

- <https://regex101.com> and <http://regexpr.com> - websites to test Regex using different programming languages
- <https://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher.html> - a quick reference for Regex from Oracle
- <http://regexone.com> - interactive tutorials for Regex
- <http://www.regular-expressions.info/tutorial.html> - a comprehensive tutorial on regular expressions

- **Regular expressions** describe **patterns** for searching through text
- Define **special characters, operators** and **constructs** for building complex pattern
- Can utilize **character classes, groups, quantifiers** and more



# Questions?



# SoftUni Diamond Partners

**SUPER  
HOSTING  
.BG**



**Coca-Cola HBC  
Bulgaria**

 **Flutter**<sup>TM</sup>  
International

**INDEAVR**  
Serving the high achievers



**AMBITIONED**

 **DRAFT  
KINGS**



**BOSCH**

 **Postbank**  
*Решения за твоето утре*

 **PHAR  
VISION**



**SmartIT**

**DXC**  
TECHNOLOGY

**createX**

- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](http://softuni.bg), [about.softuni.bg](http://about.softuni.bg)

- Software University Foundation

- [softuni.foundation](http://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)

- Software University Forums

- [forum.softuni.bg](http://forum.softuni.bg)



Software University



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

