

CS 162FZ: Introduction to Computer Science II

Lecture 04

Regular Expressions

Dr. Chun-Yang Zhang

What is a regular expression?

- A **regular expression** is a special sequence of characters that helps you **match** or **find** other strings or sets of strings, using a specialized **syntax** held in a **pattern**.
 - They can be used to search, edit, or manipulate text and data.
-

Regular Expressions: RegEx

- **Regular Expressions** are routinely used for **validating** input and **retrieving** information
- They ensure some level of validity when users are required to input (including web forms):
 - ❑ *peoples name, postal address, birth dates, Zip code, phone numbers, , credit card numbers*
 - ❑ *user ID names, passwords (& levels of security), email addresses, WWW address, file names*
 - ❑ *product codes, valid dates, cash amounts, MU Moodle course codes ...*

RegEx can find or specify:

- We can find/specify any of the following:
- The word "car" when it appears as an isolated word
- The sequence of characters "car" appearing **consecutively** in any context, such as in "**car**", "**cartoon**", or "**bicarbonate**"
- The characters "car" occurring in that order with other characters between them, such as in "**Icelander**" or "**chandler**"

RegEx can find or specify:

- The word "car" when **preceded** by the word "blue" or "red"
- The word "car" when **notpreceded** by the word "motor"
- A Euro sign immediately followed by one or more digits, and then optionally a period and exactly two more digits (for example, "€100" or "€245.99").

Regular Expressions (Regex)

- A concise means of matching strings
 - Written in a formal language that is **VERY** widely used
 - Java, C, C++, .Net, Python, Perl, Ruby, Tcl, Unix (grep), and many text editors...
 - See Java's `Pattern` class.
 - `matches()`, but also applies to `replace()` and `split()`
-

Password Example

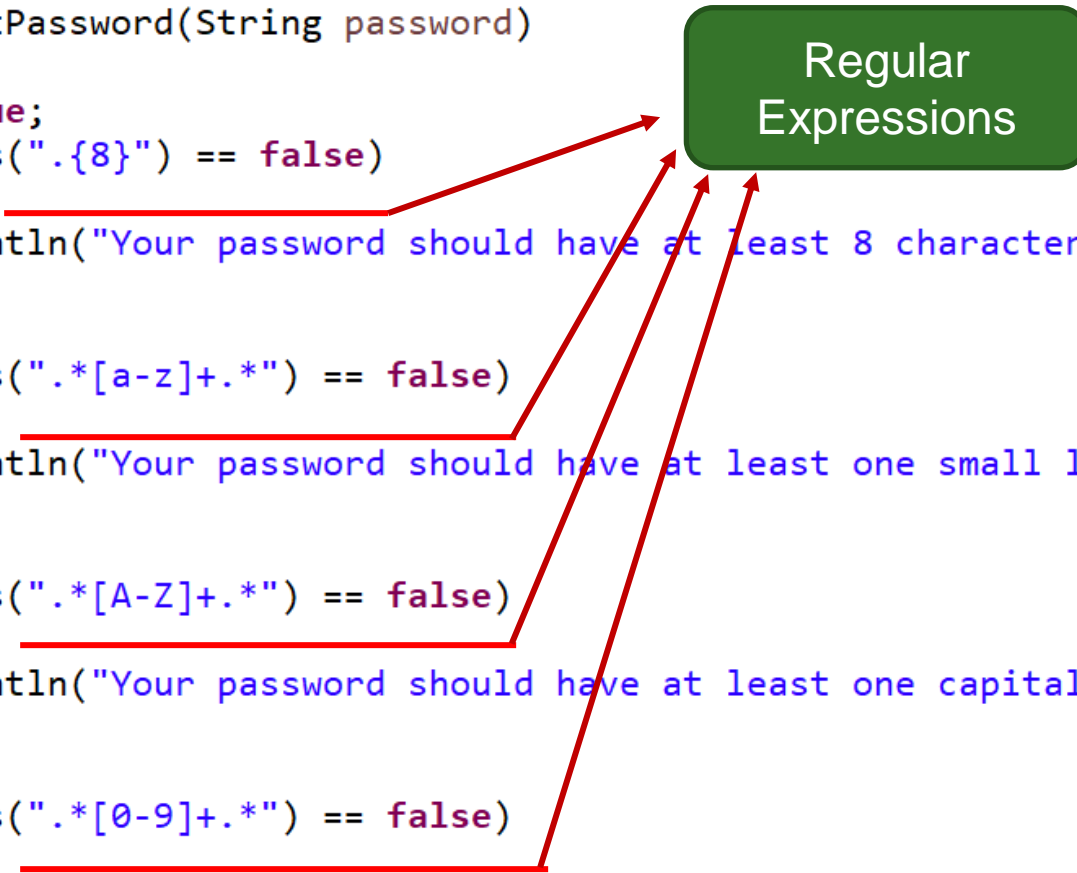
- Write a Java method called testPassword that accepts a single string variable called password.
 - Your method should examine this password and print suggestions to the user about how to improve the quality of that password.
 - Use **regular expressions** to examine the password.
-

Good Passwords

- A good password has 8 characters or more
- A good password contains at least one lower case letter
- A good password contains at least one capital letter
- A good password contains at least one digit

Code Segment of Password Example

```
public static void testPassword(String password)
{
    boolean correct=true;
    if(password.matches(".{8}") == false)
    {
        System.out.println("Your password should have at least 8 characters");
        correct=false;
    }
    if(password.matches(".*[a-z]+.*") == false)
    {
        System.out.println("Your password should have at least one small letter");
        correct=false;
    }
    if(password.matches(".*[A-Z]+.*") == false)
    {
        System.out.println("Your password should have at least one capital letter");
        correct=false;
    }
    if(password.matches(".*[0-9]+.*") == false)
    {
        System.out.println("Your password should have at least one digit");
        correct=false;
    }
}
```



Regular Expressions

String.matches(String regex)

- Regular expressions are used by both the
 - String class using matches() method, and the
 - Pattern & Matcher classes

■ boolean bool =
"abbbb".matches("ab*")

**Ordinary
String**

**Regular
expression**

- The general format for the matches method is
"Problem-string".matches("RegEx")

Example .matches

- `"abc".matches("abc")`
- `"Abc".matches("Abc")`
- `"Abc".matches("abc") // false`
- `"aabbaba".matches("[ab]*")`
- `"aabbaba".matches("a[ab]*")`
- `"bbaba".matches("a[ab]*") // false`

RegEx as a String Template

- We can think of the regular expression grammar as a ***template*** against which to match strings
- Typically, we either
 - ❑ find something that matches a template from a large volume of data
 - ❑ Ensure that a specific datum matches a template
 - Searching the internet, text of books, textual records ...
 - ❑ Looking for valid phone numbers, student ID, names, variables ...

Regular Expression Grammar

- **Sequence** (*and*) is the 1st character followed by 2nd character...
 - ❑ abc // exact match
 - ❑ Abc // case sensitive
- **Alternatives** (*or*) are enclosed in []
 - ❑ ca[bdn] // cab, cad, can
//NOT car, cat...
 - ❑ Diarm[au]id //alternative spellings
 - ❑ [Dd]ean //possible captials

More RegEx

- **Not** [^]

- ca[^brt] can and cad, but **not** car, cat or cab

Note ^ can be used to match the beginning of a line.

- **Ranges**

- [a-z] any lower case letter
- [A-Z] any capital letter
- [0-9] any digit
- [a-z&&[^xyz]] a-z but **Not** x, y or z

Quantifiers

- * **zero or more** times (Kleene *)
 - ❑ ab^* a ab abb abbbb
 - ❑ $[ab]^*$ aa aabbababba aabbaba abba
bbba
- + **One or more** times
 - ❑ ab^+ ab abb abbbb
 - ❑ $[0-9]^+$ any sequence of >1 digits
 - ❑ $[A-Z]^+$ any sequence of >1 capitals
- ? **Zero or once** (An Optional character)
 - ❑ $Colou?r$ Colour Color
 - ❑ $rea?d$ red read, but **not** reed

Counted Items

- Counted number of items
 - $x\{3\}$ // xxx only
- At least number of items
 - $x\{3,\}$ /// xxx xxxx xxxxxxxx etc *3 or more*
- Between 2 and 4 instances of
 - $x\{2,4\}$ xx, xxx and xxxx only
 - $.[a-z]\{2,4\}$ Top level of email address
(.ie .com .info)

Wild Character = “ . ”

- The dot . is the wild character which allows for any character in a string except the new line character

For example:

re.d // matches read reed rezd

Special Characters

- Special characters are occasionally used within “complicated” strings
- How do we get a String that contains the ” character?
- We use **backslash** so that the following character is not treated in the usual way
 - ❑ “the quote \” mark “ *matches the quote “ mark*
- So how do we get a backslash in a string?
 - ❑ “one \\ character” **Matches** *one backslash \ character*
 - ❑ “two backslash \\\ characters” **Matches** *two backslash \\ characters*

Special Characters & Strings

- Write down the Java code that **creates** the following Strings
- **What Java sees What i must type**
- “ab”c “ab\”c”
- “ab””c” “ab\”\”c”
- “ab\c” “ab\\c”
- “ab\\c” “ab\\\\c”

Special Characters in RegEx

- Remember the `.` will match any character
- Note: the backslash `\` character is a special character. It means, **do Not** treat the following character in the normal way
 - `\.` The full stop character
 - `\b` word boundary
 - `\s` white space (space or tab)
 - `\\` the backslash `\` character
 - `\t` the Tab character

RegEx Problems

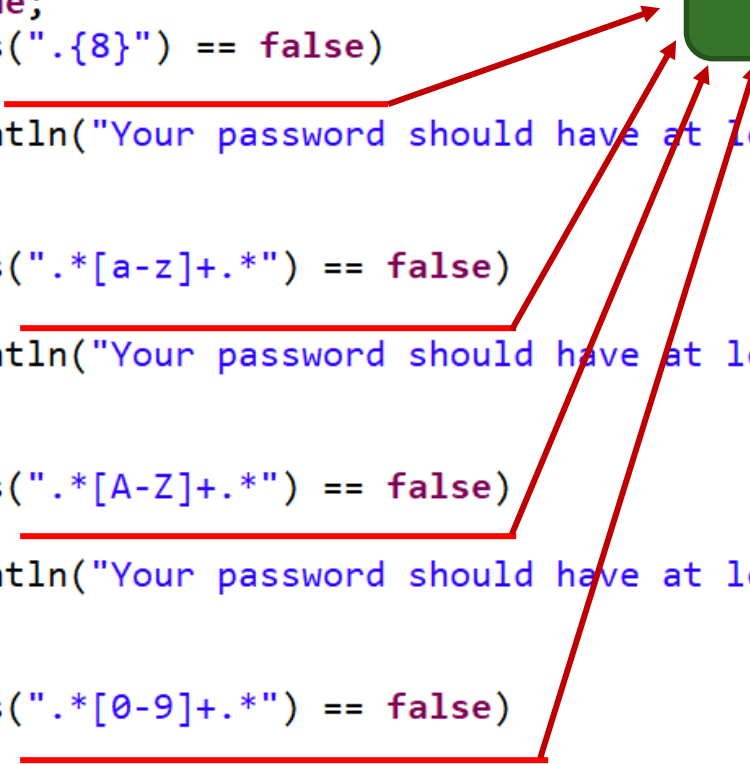
- Specify a valid variable name
 - ❑ `"abc12".matches("[a-z]+[a-z1-9]*")`
- Specify a forename followed by surname (lowercase)
 - ❑ `"tom jones".matches("[a-z]*\\s[a-z]*")`
- Can you specify
 - ❑ A valid email address?
 - ❑ Are there any limitations to your answer?
 - ❑ A dollar amount?

Specify a RegEx for the

- NUIM course codes
 - ❑ *CS161, CS162 etc*
- A full Euro amount
 - ❑ *€50, €995, €65000*
- Roman numerals (between I and VII)
 - ❑ *I, II, III, IV, V, VI, VII,*
- Any Roman numeral
 - ❑ Using the characters `IVXCLM`

Password Example Revist

```
public static void testPassword(String password)
{
    boolean correct=true;
    if(password.matches(".{8}") == false)
    {
        System.out.println("Your password should have at least 8 characters");
        correct=false;
    }
    if(password.matches(".*[a-z]+.*") == false)
    {
        System.out.println("Your password should have at least one small letter");
        correct=false;
    }
    if(password.matches(".*[A-Z]+.*") == false)
    {
        System.out.println("Your password should have at least one capital letter");
        correct=false;
    }
    if(password.matches(".*[0-9]+.*") == false)
    {
        System.out.println("Your password should have at least one digit");
        correct=false;
    }
}
```



Regular Expressions

java.util.regex package

Pattern Class – A Pattern object is a compiled representation of a regular expression. The Pattern class provides no public constructors. To create a pattern, you must first invoke one of its public static compile() methods, which will then return a Pattern object. These methods accept a regular expression as the first argument.

java.util.regex package

Matcher Class – A Matcher object is the engine that interprets the pattern and performs match operations against an input string. Like the Pattern class, Matcher defines no public constructors. You obtain a Matcher object by invoking the `matcher()` method on a Pattern object.

java.util.regex package

PatternSyntaxException –

A PatternSyntaxException object is an unchecked exception that indicates a syntax error in a regular expression pattern.

Capturing Groups

- **Capturing groups** are a way to treat multiple characters as a single unit. They are created by placing the characters to be grouped inside a set of parentheses. For example, the regular expression `(dog)` creates a single group containing the letters "d", "o", and "g".
- Capturing groups are numbered by counting their opening parentheses from the left to the right. In the expression `((A)(B(C)))`, for example, there are four such groups –
`((A)(B(C)))`
`(A)`
`(B(C))`
`(C)`

groupCount

- To find out how many groups are present in the expression, call the groupCount method on a matcher object. The groupCount method returns an **int** showing the number of capturing groups present in the matcher's pattern.
 - There is also a special group, group 0, which always represents the entire expression. This group is not included in the total reported by groupCount.
-

groupCount - Example

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RegexMatches {

    public static void main( String args[] ) {
        // String to be scanned to find the pattern.
        String line = "This order was placed for QT3000! OK?";
        String pattern = "(.*)(\\d+)(.*)";

        // Create a Pattern object
        Pattern r = Pattern.compile(pattern);

        // Now create matcher object.
        Matcher m = r.matcher(line);
        if (m.find( )) {
            System.out.println("Found value: " + m.group(0) );
            System.out.println("Found value: " + m.group(1) );
            System.out.println("Found value: " + m.group(2) );
        }else {
            System.out.println("NO MATCH");
        }
    }
}
```

Found value: This order was placed for QT3000! OK?
Found value: This order was placed for QT300
Found value: 0

Regular Expression Syntax

Subexpression	Matches
<code>^</code>	Matches the beginning of the line.
<code>\$</code>	Matches the end of the line.
<code>.</code>	Matches any single character except newline. Using m option allows it to match the newline as well.
<code>[...]</code>	Matches any single character in brackets.
<code>[^...]</code>	Matches any single character not in brackets.
<code>\A</code>	Beginning of the entire string.
<code>\Z</code>	End of the entire string.
<code>\z</code>	End of the entire string except allowable final line terminator.
<code>re*</code>	Matches 0 or more occurrences of the preceding expression.
<code>re+</code>	Matches 1 or more of the previous thing.
<code>re?</code>	Matches 0 or 1 occurrence of the preceding expression.
<code>re{n}</code>	Matches exactly n number of occurrences of the preceding expression.

Regular Expression Syntax

<code>re{ n,}</code>	Matches <code>n</code> or more occurrences of the preceding expression.
<code>re{ n, m}</code>	Matches at least <code>n</code> and at most <code>m</code> occurrences of the preceding expression.
<code>a b</code>	Matches either <code>a</code> or <code>b</code> .
<code>(re)</code>	Groups regular expressions and remembers the matched text.
<code>(?: re)</code>	Groups regular expressions without remembering the matched text.
<code>(?> re)</code>	Matches the independent pattern without backtracking.
<code>\w</code>	Matches the word characters.
<code>\W</code>	Matches the nonword characters.
<code>\s</code>	Matches the whitespace. Equivalent to <code>[\t\n\r\f]</code> .
<code>\S</code>	Matches the nonwhitespace.
<code>\d</code>	Matches the digits. Equivalent to <code>[0-9]</code> .
<code>\D</code>	Matches the nondigits.

Regular Expression Syntax

<code>\A</code>	Matches the beginning of the string.
<code>\Z</code>	Matches the end of the string. If a newline exists, it matches just before newline.
<code>\z</code>	Matches the end of the string.
<code>\G</code>	Matches the point where the last match finished.
<code>\n</code>	Back-reference to capture group number "n".
<code>\b</code>	Matches the word boundaries when outside the brackets. Matches the backspace (0x08) when inside the brackets.
<code>\B</code>	Matches the nonword boundaries.
<code>\n, \t, etc.</code>	Matches newlines, carriage returns, tabs, etc.
<code>\Q</code>	Escape (quote) all characters up to <code>\E</code> .
<code>\E</code>	Ends quoting begun with <code>\Q</code> .

The start and end Methods

```
package Lecture04;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexMatches02 {

    private static final String REGEX = "\\bcat\\b";
    private static final String INPUT = "cat cat cat cattie cat";

    public static void main( String args[] ) {
        Pattern p = Pattern.compile(REGEX);
        Matcher m = p.matcher(INPUT);    // get a matcher object
        int count = 0;

        while(m.find()) {
            count++;
            System.out.println("Match number "+count);
            System.out.println("start(): "+m.start());
            System.out.println("end(): "+m.end());
        }
    }
}
```

The matches and lookingAt Methods

```
package Lecture04;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexMatches03 {

    private static final String REGEX = "foo";
    private static final String INPUT = "fooooooooooooooooo";
    private static Pattern pattern;
    private static Matcher matcher;

    public static void main( String args[] ) {
        pattern = Pattern.compile(REGEX);
        matcher = pattern.matcher(INPUT);

        System.out.println("Current REGEX is: "+REGEX);
        System.out.println("Current INPUT is: "+INPUT);

        System.out.println("lookingAt(): "+matcher.lookingAt());
        System.out.println("matches(): "+matcher.matches());
    }
}
```

replaceFirst and replaceAll Methods

```
package Lecture04;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexMatches04 {

    private static String REGEX = "dog";
    private static String INPUT = "The dog says meow. " + "All dogs say meow.";
    private static String REPLACE = "cat";

    public static void main(String[] args) {
        Pattern p = Pattern.compile(REGEX);

        // get a matcher object
        Matcher m = p.matcher(INPUT);
        INPUT = m.replaceAll(REPLACE);
        System.out.println(INPUT);
    }
}
```

The appendReplacement and appendTail Methods

```
package Lecture04;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexMatches05 {

    private static String REGEX = "a*b";
    private static String INPUT = "aabfooaabfooabfoob";
    private static String REPLACE = "-";
    public static void main(String[] args) {

        Pattern p = Pattern.compile(REGEX);

        // get a matcher object
        Matcher m = p.matcher(INPUT);
        StringBuffer sb = new StringBuffer();
        while(m.find()) {
            m.appendReplacement(sb, REPLACE);
        }
        m.appendTail(sb);
        System.out.println(sb.toString());
    }
}
```

PatternSyntaxException Class Methods

Sr.No.	Method & Description
1	public String getDescription() Retrieves the description of the error.
2	public int getIndex() Retrieves the error index.
3	public String getPattern() Retrieves the erroneous regular expression pattern.
4	public String getMessage() Returns a multi-line string containing the description of the syntax error and its index, the erroneous regular expression pattern, and a visual indication of the error index within the pattern.

Eliza - ChatBot



- A famous 1960's computer program that acts like a Rogerian psychotherapist, based on simple pattern matching.
- Eliza uses a kind of RegEx string matching approach.

```
File Edit Options Buffers Tools Help
http://nlp-addiction.com/eliza/

I am the psychotherapist. Please, describe your problems.
Each time
you are finished talking, type RET twice.

I have a terrible headache.

Maybe you should consult a doctor of medicine, I am a psychiatrist.

Okay, I'll do that then.

Can you elaborate on that?

What are you talking about? No I cannot elaborate. See you later.

Why do you say that?

Now you're just talking nonsense!

What makes you believe now I am just talking nonsense?

You are not making any sense.

Perhaps your plans have something to do with this.
-u: ** *doctor* (Doctor Fill)--L1--Top-----
IX
```

Chat Bot

- Eliza uses regular expressions operating on sentences (rather than individual characters)
- Find the RegEx *“I feel X”* and reply with the RegEx
 - ❑ *“Do you often feel X?”*
- *“I am X”*
 - ❑ *“Do you believe it is normal to be X”*

Conclusion

- RegEx are used throughout computer science
- We can easily specify a template for a valid string
- Special characters cause complication
 - ❑ \ for strings, \\ for RegEx themselves
- The famous Eliza ChatBot uses a similar approach

Test a RegEx:
<https://regex101.com/>

Further reading
www.regular-expressions.info

Test a RegEx in Java
<https://www.regexplanet.com/advanced/java/index.html>

More Regex

Generate Reg Ex to :

- Accept the words *Ireland* and *Iceland*
- Accept only the words *dart*, *dark*, *darf*
- Accept the words *ward*, *Ward*, *card*, *Card*, *lard*, *Lard*
- Accept any word beginning with *a* *c* or *d* or *l* that is 4 letters long and has *ar* as the 2nd and 3rd letter
- Accept any sequence of 4 characters that has a digit in the first and last positions and letters everywhere else.
- What does the following REg Ex accept? - *[vcr]at*
- *vcr*, *vcrat*, *vat*, *at*,