



CS472 WAP

Regular Expressions

JavaScript Functional Programming

Except where otherwise noted, the contents of this document are Copyright 2012 Marty Stepp, Jessica Miller, Victoria Kirst and Roy McElmurry IV. All rights reserved. Any redistribution, reproduction, transmission, or storage of part or all of the contents in any form is prohibited without the author's expressed written permission. Slides have been modified for Maharishi University of Management Computer Science course CS472 in accordance with instructors agreement with authors.

**Maharishi International University Fairfield,
Iowa** © 2020



All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi International University.

Main Point Preview

- Regular Expressions are an industry standard way of performing pattern matching. Basically every language has support for them (including HTML5, which isn't even a programming language). Pattern matching is often used for input validation.
- *A pattern can describe many different permutations on the same theme. It's an abstraction that lets you check if something is correct; helping you keep input clean. Purification leads to Progress. One of the practical benefit of TM is the removal of stress.*

JavaScript Regular Expressions



- You can create a regular expression in two ways
 - Either using the RegExp constructor
 - Or with the a regular expression literal (which starts and ends with a /)
 - IDEs often have additional syntax highlighting support for this

```
let emailMatch1 = new RegExp("^ [a-zA-Z_\\-]+@ ( ( [a-zA-Z_\\-] )+\\. )+ [a-zA-Z] {2,} $" );  
let emailMatch2 = /^[a-zA-Z_\\-]+@ ( ( [a-zA-Z_\\-] )+\\. )+ [a-zA-Z] {2,} $/;
```

- The most common thing to do is to see if a string matches the pattern
 - Either by using the .match() method on the string
 - Or the .test() on the regex

```
"mzijlstra@miu.edu".match(emailMatch2) // boolean value  
emailMatch1.test("mzijlstra@miu.edu") // same boolean value
```

<input> pattern attribute



The **pattern** attribute specifies a regular expression that the **<input>** element's value is checked against. The **pattern** uses the ECMAScript (i.e. JavaScript) flavor of regex.

Note: The **pattern** attribute works with the following input types: **text**, **date**, **search**, **url**, **tel**, **email**, and **password**.

Tip: Use the global **title** attribute to describe the pattern to help the user.

```
<form action="demo_form.jsp">
```

Country code:

```
<input type="text" name="country_code" pattern="[A-Za-z]{3}" title="Three letter country code ">
```

```
<input type="submit">
</form>
```

Country code:

! Please match the requested format.
Three letter country code

Regular expressions

`^[a-zA-Z_\-]+@([a-zA-Z_\-]+\.)+[a-zA-Z]{2,4}$`

- ▶ Test whether a string matches the expression's pattern
- ▶ powerful but tough to read
 - ▶ (the above regular expression matches email addresses)
- ▶ used in all languages:
 - ▶ Java, PHP ,JavaScript, HTML, C#, and other languages
- ▶ Many IDEs allow regexes in search/replace



Basic regular expressions

The simplest regexes simply matches any string that contains that text.

abc

above regular expression matches any string containing "abc":

- ▶ YES: "abc", "abcdef", "defabc", " .=.abc.=.", ...
- ▶ NO: " ABC" , " fedcba", "ab c", "PHP", ...
- ▶ Note that html5 <input pattern='abc' .. >has implicit anchors ^ and \$, so abc is really ^abc\$
- ▶ Regular expressions are case-sensitive by default.

Wildcards

A dot `.` matches exactly **one-character** except a `\n` line break

`.oo.y` matches "Doocy", "goofy", "LooNy", ...

Special characters: |, (), \

| means OR

abc | def | g matches "abc", "def", or "g"

() are for grouping

(Homer | Marge) Simpson

matches "Homer Simpson" or "Marge Simpson"

\ escapes a special character

many characters must be escaped to match them literally: / \ \$. [] () ^ * + ?

<br \ / > matches lines containing
 tags

Quantifiers: *, +, ?

***** means 0 or more occurrences

abc* matches "ab", "abc", "abcc", "abccc", ...

a(bc)* matches "a", "abc", "abcbc", "abcbcbc", ...

a.*a matches "aa", "aba", "a8qa", "a!?xyz__9a", ...

+ means 1 or more occurrences

a(bc)+ matches "abc", "abcbc", "abcbcbc", ...

Goo+gle matches "Google", "Goooogle", "Goooooogle", ...

? means 0 or 1 occurrences

a(bc)? matches "a" or "abc"

More quantifiers: {min,max}

{min,max} means between min and max occurrences (inclusive)

a(bc){2,4} matches "abcbc", "abcbcbc", or "abcbcbcbc"

min or **max** may be omitted to specify any number

{2,} means 2 or more

{,6} means up to 6

{3} means exactly 3

Anchors: ^ and \$

^ represents the beginning of the string or line;

\$ represents the end

Jess matches all strings that contain Jess;

^Jess matches all strings that start with Jess;

Jess\$ matches all strings that end with Jess;

^Jess\$ matches the exact string "Jess" only

^Mart.*Stepp\$ matches "MartStepp", "Marty Stepp", "Martin D Stepp", ... but NOT "Marty Stepp stinks" or "I H8 Martin Stepp"

The html5 spec states that ^ and \$ are implicit

Character sets: []

[] group characters into a character set, will match any **single character** from the set

[bcd]art matches strings containing "bart",
"cart", and "dart"

equivalent to **(b|c|d)art** but shorter

inside [], many of the modifier keys act as normal characters

what[!*?]* matches "what", "what!", "what?*!", "what??!", ...

What regular expression matches DNA (strings of A, C, G, or T)?

[ACGT]+

Character ranges: [start-end]

inside a character set, specify a range of characters with -

`[a-z]` matches any lowercase letter

`[a-zA-Z0-9]` matches any lower- or uppercase letter or digit

an initial `^` inside a character set negates it

`[^abcd]` matches any character other than a, b, c, or d

inside a character set, - must be escaped to be matched

`[+\-]?[0-9]+` matches an optional + or -, followed by at least one digit

What regular expression matches letter grades such as A, B+, or D- ?

`[A-Z][+\-]?`

Escape sequences

Special escape sequence character sets:

\d matches any digit (same as [0-9])

\D any non-digit ([^0-9])

\w matches any word character (same as [a-zA-Z_0-9])

\W any non-word char

\s matches any whitespace character (, \t, \n, etc.)

\S any non-whitespace

What regular expression matches dollar amounts of at least \$100.00 ?

\\$[1-9]\d{2,}\.\d{2}

Example - URL



- ▶ An `<input>` element with `type="url"` that must start with `http://` or `https://` followed by at least one character:

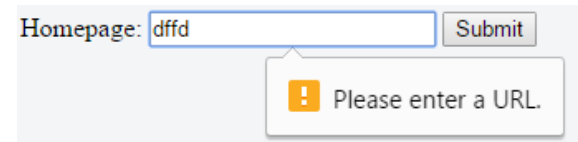
```
<form action="demo_form.jsp">
```

Homepage :

```
<input type="url" name="website" pattern="https?: /\.+"
title="Please enter a URL">
```

```
<input type="submit">
```

```
</form>
```



The screenshot shows a web form with the label "Homepage:" followed by a text input field containing the text "dffd". To the right of the input field is a "Submit" button. Below the input field, a validation error message is displayed in a white box with a grey border. The message starts with an orange exclamation mark icon and contains the text "Please enter a URL.".

Main Point

- Regular Expressions are an industry standard way of performing pattern matching. Basically every language has support for them (including HTML5, which isn't even a programming language). Pattern matching is often used for input validation.
- A pattern can describe many different permutations on the same theme. It's an abstraction that lets you check if something is correct; helping you keep input clean. Purification leads to Progress. One of the practical benefit of TM is the removal of stress.



Example - email



► An `<input>` element with `type="email"` that must be in the following order: characters@characters.domain

(characters followed by an @ sign, followed by more characters, and then a "." and then 2 or 3 letters)

```
<form action="demo_form.jsp">
```

E-mail:

```
<input type="email" name="email" pattern="[a-z0-9._+\\-]+@[a-z0-9.\\-]+\\. [a-z]{2,3}">
```

```
<input type="submit">
```

```
</form>
```

E-mail:

! Please include an '@' in the email address. 'fdfd' is missing an '@'.

Examples - password



- ▶ An `<input>` element with `type="password"` that must contain 8 or more characters that are of at least one number, and one uppercase and lowercase letter:

```
<form action="demo_form.jsp">
```

Password:

```
<input type="password" name="pw" pattern="(=?\d+) (=?[a-z]*) (=?[A-Z]*) .  
{8,}" title="Must contain at least one number and one uppercase and  
lowercase letter, and at least 8 or more ch
```

```
<input type="submit">
```

```
</form>
```

Password:

! Please match the requested format.

Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters

Advanced: Lookaround

Positive lookahead **(?=A) B**

Once a group starts with **?=** it means positive lookahead. Find expression A first, if found then expression B follows.

Negative lookahead **(?!A) B**

Once a group starts with **?!** it means negative lookahead.

First check if expression A is not found, then check if expression B follows.

Example - Search



- ▶ An <input> element with type="search" that CANNOT contain the following characters: ' or “

```
<form action="demo_form.jsp">
```

```
  Search: <!-- x27 is a single quote and \x22 is a double quote -->
```

```
  <input type="search" name="search" pattern="^[^\x27\x22]+" title="cannot  
contain quote characters">
```

```
  <input type="submit">
```

```
</form>
```

Search: Submit

! Please match the requested format.
Invalid input