

JavaScript Workshop

Regular Expressions

Regular expressions

- Regular expressions provide a means of defining a pattern of characters, which can be used to search or replace characters in a string where they fit the defined pattern.
- Regular expressions in JavaScript use the RegExp object.

There are two ways of creating a RegExp object:

```
var myRegExp = /\b/'\b/; // creating a string literal
```

```
var myRegExp = new RegExp("/\b/'\b/"); // instantiating a new object
```

Simple regular expressions

- Defining patterns of characters using regular expression syntax can get very complex, so simple expressions will be used.

```
var myString = "Paul, Paula, Pauline, paul, Paul";
```

We want to replace any occurrence of the name "Paul" with "Ringo". The pattern is Paul, the forward slashes (/) mark the start and end of an expression.

```
var myRegExp = /Paul/;
```

```
myString = myString.replace(myRegExp, "Ringo");
```

Global matching

- By default the RegExp object looks for the first matching pattern then stops. What we want is a global match, which is a search for all possible matches to be made and replaced. To do this, there are attributes that can be defined with a pattern.

g - Global match. Looks for all matches of the pattern.

i - Pattern is case-insensitive. Paul and paul are now the same.

```
var myRegExp = /Paul/gi;
```

```
myString = myString.replace(myRegExp, "Ringo");
```

Special characters

\d - Any digit from 0 to 9

\D - Any character that is not a digit

\w - Any word character; that is A-Z, a-z, 0-9, and underscore

\W - Any non-word character

\s - Any white space: including tab, new line, carriage return, etc.

\S - Any non-space character

. - Any single character other than the new line character *\n*

[...] - Any one character between the brackets

[^...] - Any one character, but not one of those inside the brackets

Repetition characters

- Repetition characters specify how many of the last item or character we want to match:

{n} - Match *n* of the previous item

{n,} - Match *n* or more of the previous item

{n,m} - Match at least *n* and at most *m* of the preceding item

? - Match the previous item zero or one time

+ - Match the previous item one or more times

*** - Match the previous item zero or more times

Position characters

- Position characters allow you to specify either where the match should start or end or what will be on either side of the character pattern.

^ - The pattern must be at the start of the string

\$ - The pattern must be at the end of the string

RegExp methods

- The `test()` method returns true if the given string (the only argument) matches the pattern and false if not:

```
var sToMatch = "cat";  
var reCat = /cat/;  
alert(reCat.test(sToMatch)); // outputs true
```

- The `exec()` method takes a string as an argument, returns an array containing all matches:

```
var sToMatch = "a bat, a Cat, a fAt baT, a faT cat";  
var reAt = /at/;  
var arrMatches = reAt.exec(sToMatch);
```

Validate an email address

- One of the most common uses for regular expressions is validating an email address.

```
var reg_exp = /^[a-z][\w\.]*@[\w\.]+\.[a-z]{2,3}/i;
```

Step 1: Checks for a non number on the first character: `^[a-z]`

Step 2: Remaining before @ can be any word character: `[\w\.]`*

Step 3: Address must have @ sign: `@`

Step 4: To right of @, is domain name, any characters: `[\w\.]+`

Step 5: Domain suffix, a dot(.) followed by 2 or 3 letters: `\.[a-z]{2,3}`

Step 6: Disregard case: `/i`

- JavaScript Workshop website:
<http://www.javascriptworkshop.com>
- Any questions?
frank@javascriptworkshop.com - Frank Stepanski
- Recommended reading:
[JavaScript: A Beginner's Guide, Second Edition](#)