http://www.javascriptkit.com/spacer.gif

#### http://www.javascriptkit.com/jkincludes/arrow2.gifCategories of Pattern Matching Characters

Pattern-matching characters can be grouped into various categories, which will be explained in detail later. By understanding these characters, you understand the language needed to create a regular expression pattern. The categories are:

* **Position matching**- You wish to match a substring that occurs at a specific location within the larger string. For example, a substring that occurs at the very beginning or end of string.
* **Special literal character matching**- All alphabetic and numeric characters by default match themselves literally in regular expressions. However, if you wish to match say a newline in Regular Expressions, a special syntax is needed, specifically, a backslash (\) followed by a designated character. For example, to match a newline, the syntax "\n" is used, while "\r" matches a carriage return.
* **Character classes matching**- Individual characters can be combined into character classes to form more complex matches, by placing them in designated containers such as a square bracket. For example, /[abc]/ matches "a", "b", or "c", while /[a-zA-Z0-9]/ matches all alphanumeric characters.
* **Repetition matching**- You wish to match character(s) that occurs in certain repetition. For example, to match "555", the easy way is to use /5{3}/
* **Alternation and grouping matching**- You wish to group characters to be considered as a single entity or add an "OR" logic to your pattern matching.
* **Back reference matching**- You wish to refer back to a subexpression in the same regular expression to perform matches where one match is based on the result of an earlier match.

The following are categorized tables explaining the above:

#### http://www.javascriptkit.com/jkincludes/arrow2.gifPosition Matching

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| ^ | Only matches the beginning of a string. | /^The/ matches "The" in "The night" by not "In The Night" |
| $ | Only matches the end of a string. | /and$/ matches "and" in "Land" but not "landing" |
| \b | Matches any word boundary (test characters must exist at the beginning or end of a word within the string) | /ly\b/ matches "ly" in "This is really cool." |
| \B | Matches any non-word boundary. | /\Bor/ matches “or” in "normal" but not "origami." |

#### http://www.javascriptkit.com/jkincludes/arrow2.gifLiterals

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| Alphanumeric | All alphabetical and numerical characters match themselves literally. So /2 days/ will match "2 days" inside a string. |
| \n | Matches a new line character |
| \f | Matches a form feed character |
| \r | Matches carriage return character |
| \t | Matches a horizontal tab character |
| \v | Matches a vertical tab character |
| \xxx | Matches the ASCII character expressed by the octal number xxx.  "\50" matches left parentheses character "(" |
| \xdd | Matches the ASCII character expressed by the hex number dd.  "\x28" matches left parentheses character "(" |
| \uxxxx | Matches the ASCII character expressed by the UNICODE xxxx.  "\u00A3" matches "£". |

The backslash (\) is also used when you wish to match a special character literally. For example, if you wish to match the symbol "$" literally instead of have it signal the end of the string, backslash it: /\$/

#### http://www.javascriptkit.com/jkincludes/arrow2.gifCharacter Classes

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| [xyz] | Match any one character enclosed in the character set. You may use a hyphen to denote range. For example. /[a-z]/ matches any letter in the alphabet, /[0-9]/ any single digit. | /[AN]BC/ matches "ABC" and "NBC" but not "BBC" since the leading “B” is not in the set. |
| [^xyz] | Match any one character not enclosed in the character set. The caret indicates that none of the characters  **NOTE:** the caret used within a character class is not to be confused with the caret that denotes the beginning of a string. Negation is only performed within the square brackets. | /[^AN]BC/ matches "BBC" but not "ABC" or "NBC". |
| **.** | (Dot). Match any character except newline or another Unicode line terminator. | /b.t/ matches "bat", "bit", "bet" and so on. |
| \w | Match any alphanumeric character including the underscore. Equivalent to [a-zA-Z0-9\_]. | /\w/ matches "200" in "200%" |
| \W | Match any single non-word character. Equivalent to [^a-zA-Z0-9\_]. | /\W/ matches "%" in "200%" |
| \d | Match any single digit. Equivalent to [0-9]. |  |
| \D | Match any non-digit. Equivalent to [^0-9]. | /\D/ matches "No" in "No 342222" |
| \s | Match any single space character. Equivalent to [ \t\r\n\v\f]. |  |
| \S | Match any single non-space character. Equivalent to [^ \t\r\n\v\f]. |  |

#### http://www.javascriptkit.com/jkincludes/arrow2.gifRepetition

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| {x} | Match exactly x occurrences of a regular expression. | /\d{5}/ matches 5 digits. |
| {x,} | Match x or more occurrences of a regular expression. | /\s{2,}/ matches at least 2 whitespace characters. |
| {x,y} | Matches x to y number of occurrences of a regular expression. | /\d{2,4}/ matches at least 2 but no more than 4 digits. |
| ? | Match zero or one occurrences. Equivalent to {0,1}. | /a\s?b/ matches "ab" or "a b". |
| \* | Match zero or more occurrences. Equivalent to {0,}. | /we\*/ matches "w" in "why" and "wee" in "between", but nothing in "bad" |
| + | Match one or more occurrences. Equivalent to {1,}. | /fe+d/ matches both "fed" and "feed" |

#### http://www.javascriptkit.com/jkincludes/arrow2.gifAlternation & Grouping

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| ( ) | Grouping characters together to create a clause. May be nested. | /(abc)+(def)/ matches one or more occurrences of "abc" followed by one occurrence of "def". |
| | | Alternation combines clauses into one regular expression and then matches any of the individual clauses. Similar to "OR" statement. | /(ab)|(cd)|(ef)/ matches "ab" or "cd" or "ef". |

#### http://www.javascriptkit.com/jkincludes/arrow2.gifBackreferences

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| ( )\n | Matches a parenthesized clause in the pattern string. n is the number of the clause to the left of the backreference. | (\w+)\s+\1 matches any word that occurs twice in a row, such as "hubba hubba." The \1 denotes that the first word after the space must match the portion of the string that matched the pattern in the last set of parentheses. If there were more than one set of parentheses in the pattern string you would use \2 or \3 to match the appropriate grouping to the left of the backreference. Up to 9 backreferences can be used in a pattern string. |

### http://www.javascriptkit.com/jkincludes/arrow2.gifPattern Switches

In addition to the pattern-matching characters, you can use switches to make the match global or case- insensitive or both: Switches are added to the very end of a regular expression.

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Example** |
| i | Ignore the case of characters. | /The/i matches "the" and "The" and "tHe" |
| g | Global search for all occurrences of a pattern | /ain/g matches both "ain"s in "No pain no gain", instead of just the first. |
| gi | Global search, ignore case. | /it/gi matches all "it"s in "It is our IT department" |

#### String and Regular Expression methods

The String object has four methods that take regular expressions as arguments. These are your workhorse methods that allow you to match, search, and replace a string using the flexibility of regular expressions:

#### http://www.javascriptkit.com/jkincludes/arrow2.gifString Methods Using Regular Expressions

|  |  |
| --- | --- |
| **Method** | **Description** |
| match( regular expression ) | Executes a search for a match within a string based on a regular expression. It returns an array of information or null if no match are found.  **Note:** Also updates the $1…$9 properties in the RegExp object. |
| replace( regular expression, replacement text ) | Searches and replaces the regular expression portion (match) with the replaced text instead.  **Note:** Also supports the replacement of regular expression with the specified RegExp $1…$9 properties. |
| split ( string literal or regular expression ) | Breaks up a string into an array of substrings based on a regular expression or fixed string. |
| search( regular expression ) | Tests for a match in a string. It returns the index of the match, or -1 if not found. Does NOT support global searches (ie: "g" flag not supported). |

Here are a few examples:

var string1="Peter has 8 dollars and Jane has 15"

parsestring1=string1.match(/\d+/g) //returns the array [8,15]

var string2="(304)434-5454"

parsestring2=string2.replace(/[\(\)-]/g, "") //Returns "3044345454" (removes "(", ")", and "-")

var string3="1,2, 3, 4, 5"

parsestring3=string3.split(/\s\*,\s\*/) //Returns the array ["1","2","3","4","5"]

Delving deeper, you can actually use the replace() method to **modify**- and not simply replace- a substring. This is accomplished by using the $1…$9 properties of the RegExp object. These properties are populated with the contents of the portions of the searched string that matched the portions of the search pattern contained within parentheses. The following example illustrates how to use the replace method to swap the order of first and last names and insert a comma and a space in between them:

<SCRIPT language="JavaScript1.2">

var objRegExp = /(\w+)\s(\w+)/;

var strFullName = "Jane Doe";

var strReverseName = strFullName.replace(objRegExp, "$2, $1");

alert(strReverseName) //alerts "Doe, John"

</SCRIPT>

The output of this code will be “Doe, Jane”. How this works is that the pattern in the first parentheses matches “Jane” and this string is placed in the RegExp.$1 property. The \s (space) character match is not saved to the RegExp object because it is not in parentheses. The pattern in the second set of parentheses matches “Doe” and is saved to the RegExp.$2 property. The String replace() method takes the Regular Expression object as its first argument and the replacement text as the second argument. The $2 and $1 in the replacement text are substitution variables that will substitute the contents of RegExp.$2 and RegExp.$1 in the result string.

You can also use replace() method to strip unwanted characters from a string before testing the string for validity or before saving the string to a [database](http://www.javascriptkit.com/javatutors/re3.shtml). It can be used to add formatting characters for the display of a string as well.

#### http://www.javascriptkit.com/jkincludes/arrow2.gifRegExp methods and properties

You just saw several regular expression related string methods; in most situations, they are all you need for your string manipulation needs. However, true to the versatility of regular expressions, the Regular Expression (RegExp) object itself also supports two methods that mimic the functions of their string counterparts, the difference being these two methods take strings as parameters, while with String functions, they take a RegExp instead. The following describes the methods and properties of the regular expression object.

#### http://www.javascriptkit.com/jkincludes/arrow2.gifMethods

|  |  |
| --- | --- |
| **Method** | **Description** |
| test(string) | Tests a string for pattern matches. This method returns a Boolean that indicates whether or not the specified pattern exists within the searched string. This is the most commonly used method for validation. It updates some of the properties of the parent RegExp object following a successful search. |
| exec(string) | Executes a search for a pattern within a string. If the pattern is not found, exec() returns a null value. If it finds one or more matches it returns an array of the match results. It also updates some of the properties of the parent RegExp object. |

Here is a simple example that uses test() to see if a regular expression matches against a certain string:

var pattern=/php/i

pattern.test("PHP is your friend") //returns true

#### http://www.javascriptkit.com/jkincludes/arrow2.gifRegExp instance properties

Whenever you define an instance of the regular expression (whether using the literal or constructor syntax), additional properties are exposed to this instance which you can use:

#### Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| $n | n represents a number from 1 to 9 Stores the nine most recently memorized portions of a parenthesized match pattern. For example, if the pattern used by a regular expression for the last match was /(Hello)(\s+)(world)/ and the string being searched was “Hello world” the contents of RegExp.$2 would be all of the space characters between “Hello” and “world”. |
| source | Stores a copy of the regular expression pattern. |
| global | Read-only Boolean property indicating whether the regular expression has a "g" flag. |
| ignoreCase | Read-only Boolean property indicating whether the regular expression has a "i" flag. |
| lastIndex | Stores the beginning character position of the last successful match found in the searched string. If no match was found, the lastIndex property is set to –1. |

This simple example shows how to determine whether a regular expression has the "g" flag added:

var pattern=/php/g

alert(pattern.global) //alerts true

#### Sample Usage

Now that you’ve been introduced to regular expressions and patterns, let’s look at a few examples of common validation and formatting functions.

#### - Valid Number

A valid number value should contain only an optional minus sign, followed by digits, followed by an optional dot (.) to signal decimals, and if it's present, additional digits. A regular expression to do that would look like this:

var anum=/(^-\*\d+$)|(^-\*\d+\.\d+$)/

#### - Valid Date Format

A valid short date should consist of a 2-digit month, date separator, 2-digit day, date separator, and a 4-digit year (e.g. 02/02/2000). It would be nice to allow the user to use any valid date separator character that your backend database supported such as slashes, dashes and periods. You want to be sure the user enters the same date separator character for all occurrences. The following function returns true or false depending on whether the user input matches this date format:

function checkdateformat(userinput){

var dateformat = /^\d{1,2}(\-|\/|\.)\d{1,2}\1\d{4}$/

return dateformat.test(userinput) //returns true or false depending on userinput

}

This example uses backreferencing to ensure that the second date separator matches the first one.

#### - Replace HTML tags (brackets) with entities instead

User input often times must be parsed for security or to ensure it doesn't mess up the formatting of the page. The most common task is to remove any HTML tags (brackets) entered by the user, and replace them with their entities equivalent instead. The following function does just that- replace "<" and ">" with "&lt;" and "&gt;", respectively:

function htmltoentity(userinput){

var formatted=userinput.replace(/(<)|(>)/g,

function(thematch){if (thematch=="<") return "&lt;"; else return "&gt;"})

}

The first parameter of  replace() searches for a match for either "<" **or** ">". The second parameter demonstrates something new and interesting- you can actually use a function instead of a plain replacement text as the parameter. When a function is used, the parameter of it (in this case, "thematch") contains the matched substring and returns what you wish it to be replaced with. Since we're looking to replace both "<" and ">", this function will help us return two different replacement strings accordingly.