Example on Regular Expressions

```
<!doctype html>
<html>
	<head>
		<title>regular expression demo</title>
		<script type="text/javascript">
			function validateUser()
			{
				var name=document.form1.txtname.value;
				var reg=/^[a-zA-Z]*$/;
				if(name.match(reg))
				{
					return true;
				}
				else
				{
					alert("not valid name");
					document.form1.txtname.focus();
					return false;
				}
			}
			function validateMobile()
			{
			var mobile=document.form1.txtmobile.value;
			var reg=/^\d{10}$/;
			if(mobile.match(reg))
			{
				return true;
			}
			else
			{
```

```
                    alert("invalid mobile");
                document.form1.txtmobile.focus();
                return false;

                }
                }
        function validate()
        {
            if(validateUser() && validateMobile())
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        </script>
    </head>
    <body>
        <form name="form1" onsubmit="return validate()">
            name :<input type="text" name="txtname" /><br/>
    Mobile:     <input type="text" name="txtmobile"/>  <br/>
            <input type="submit" value="submit"/>
        </form>
    </body>
</html>
```

Regular Expressions

Regular expressions are an extremely useful tool for working with text. Whether you need to validate user input, search for patterns within string

We can use Regular Expressions in any Language.

Syntax:

/^pattern$/;

/^ : represents the beginning of the pattern.

$/ : represents the ending of the pattern.

e.g.

var regex=/^[a-z]{5}$/; //allows only lower case alphabets and must have 5 chars.

var regex=/^[0-9]{4}$/; //allows only digits(0-9) and must have 4 digits

Quantifiers: These are used to specify the occurrences.

These are non-explicit quantifiers

*: which describes zero or more occurrences

+: which describes 1 or more occurrences

? : Which describes zero or 1 occurrence

e.g.

var regex=/^[a-zA-Z]*$/;

It allows only alphabets and can have either zero or more characters.

It can be blank or any string with alphabets.

var regex=/^[0-9]+$/;

It allows only digits and should have at least one digit.

var regex=/^[a-z]?$/;

It allows only alphabets (Lower case) and can have zero or 1 character.

Explicit Quantifier

{} for specifying the lower and upper bounds

It describes the minimum and maximum occurrences

e.g.

ab{2}c : abbc, aaabbccc (must have two characters)

ab{,2}c: ac, abc, abbc, aabbcc (min zero , max 2 chars)

ab{2,3}c :abbc, abbbc, aabbcc, aabbbcc (min 2, max 3)

e.g.

{n,m} : min n and max m

{,m} : min zero and max m

{n} : must have n characters

var regex=/^[a-zA-Z]{6,12}$/;

Only alphabets and between 6 to 12 length

var regex=/^[0-9]{10}$/;

Only digits and must be 10 digits

var regex=/^[0-9]{3}-[0-9]{3}-[0-9]{4}$/;

Here, first 3 digits then hypen(-) and again 3 digits ,then hypen(-) and again 4 digits.

123-222-4321

Quantifiers:

*

+

?

{n,m}

{n}

{,m}

Meta Characters

. (The period or dot operator)

It matches with any single character

^ : It is used to designate the beginning of a string

$: meta character is used to designate the end of a string

[ ]: used to specify the pattern (Character Class)

e.g.

[aeiou]

It validates any one of these characters.

[a-z]

It represents lower case alphabets (any lower case character)

[A-Z]

Upper case alphabets

[a-zA-Z]

Both alphabets

[a-zA-Z0-9]

Alpha-numerics

[012345]

0 to 5

[0-9]

All digits

[marlabs]

Any of these characters

[] : it is for defining the pattern, so that we can pick any of these characters.(others not allowed).

e.g.

/^[a-zA-Z]{6,8}$/

Any string with min 6 and max 8 characters

/^[0-9]{10}$/

Any number with 10 digits

/^\d{10}$/

Any number with 10 digits

The \ (backslash) metacharacter is used to "escape" characters

The | (pipe) metacharacter is used for alternation (conjunction)

The parentheses ( ) are used to group patterns

e.g.

/^(abc){2}[A-Z]{1}[aeiou]{5}$/

First abc 2 times

Next any one alphabet (upper case)

Next any string with 5 characters contains only ovels

abcabcAaaaaa

e.g.

. : Any single character

.* : match with either zero or more characters

Character Classes

Character classes are a mini-language within regular expressions, defined by the enclosing hard braces [ ].

The simplest character class is simply a list of characters within these braces, such as [aeiou].

It's important to note that character classes cannot be used to define words or patterns, only single characters.

To specify any numeric digit, the character class [0123456789] could be used

To specify any numeric digit using a hyphen, you would use [0-9]

Similarly for any lowercase letter, you could use [a-z], or for any uppercase letter [A-Z]

For all the alphabets: [a-zA-Z]
For all the alpha-numeric: [a-zA-Z0-9]

Note:

In the above Character class, we are specifying the pattern for alphabets by using hyphen ([a-z])

But, the hyphen is not included in that character class.

If you need a hyphen to be included in your range, specify it as the first character

For example, [-\.\?] would match any one of those four characters (note the last character is a space).

e.g.

var regex=/^[-\.\?]$/;

It allows any of these four characters.

-

.

?

Whitespace

Note:

We use backslash (\), for considering a special character in our class.

\. : allows only. (Dot)

\? : allows only?

\d: allows only digits

You can also match any character except a member of a character class by negating the class using the carat ^ as the first character in the character class.

Thus, to match any non-vowel character, you could use a character class of [^aAeEiIoOuU].

[aeiou]

It allows choosing any vowel.

[^aeiou]

Note that if you want to negate a hyphen, it should be the second character in the character class, as in [^-]

Remember that the ^ has a totally different meaning within a character class than it has at the start of a regular expression pattern.

Note:

the pattern is case sensitive.

The regular Expressions are case sensitive.

e.g.

/^b[aeiou]t$/ : bat, bet, bit, bot, but

/^[0-9]{5}$/ : 11111, 12345, 99999

/^c:\\ : c:\windows, c:\\\\\, c:\foo.txt, c:\ followed by anything else

abc$/ : abc, 123abc, any string ending with abc

(abc){2,3} : satyaabcabcmarlabs, abcabcabc [It comes any where in the string]

/^[^-][0-9]$/ : 0, 1, 2, … (Will not match -0, -1, -2, etc.)

^\d{5}$: 5 numeric digits, such as a US ZIP code.

^(\d{5})|(\d{5}-\d{4})$:5 numeric digits, or 5 digits-dash-4 digits.

^(\d{5})(-\d{4})?$: Same as previous, but more efficient.

Uses ? to make the -4 digits portion of the pattern optional, rather than requiring two separate patterns to be compared individually (via alternation).

^[+-]?\d+(\.\d+)?$: Matches any real number with optional sign.

^[+-]?\d*\.?\d*$: Same as above, but also matches empty string

/^\/*.**\/$/: Matches the contents of a C-style comment /* ..... */

/^\/\/.*$/ : matches with single line comment

Special expressions

. :Match any single character except newline

\w :Match any alphanumeric character

\s :Match any whitespace character

\d :Match any digit

\b :Match the beginning or end of a word

^ :Match the beginning of the string

$:Match the end of the string

Repetitions

*

Repeat any number of times

+

Repeat one or more times

?

Repeat zero or one time

{n}

Repeat n times

{n,m}

Repeat at least n, but no more than m times

{n,}

Repeat at least n times

\b\w{5,6}\b Find all five and six letter words

\b\d{3}\s\d{3}-\d{4} Find ten digit phone numbers

999 999-9999

\d{3}-\d{2}-\d{4} Social security number

^\w* The first word in the line or in the text

Negation

(Upper case letters)

\W : here W is upper case

Match any character that is NOT alphanumeric

\S

Match any character that is NOT whitespace

\D

Match any character that is NOT a digit

\B

Match a position that is NOT the beginning or end of a word

[^x]

Match any character that is NOT x

[^aeiou]

Match any character that is NOT one of the characters aeiou

\S+ All strings that do not contain whitespace characters

Grouping

Parentheses may be used to delimit a subexpression to allow repetition or other special treatment.

(\d{1,3}\.){3}\d{1,3} A simple IP address finder

The first part of the expression searches for a one to three digit number followed by a literal period "\.". This is enclosed in parentheses and repeated three times using the "{3}" quantifier, followed by the same expression without the trailing period.

Let's create regular Expression for E-mail:

In the email:

regex=/^\w+\@\w+\.[a-zA-Z]{2,4}$/;

Here, first any word with alpha-numeric, but at least on character

Then @ character

Then any word with at least on character

Then Character

Then a word with 2 or 4 characters

Note:

We use backslash to include the special character in the regular expression.

e.g.

\. : allows only.

\@ : allows only @

\\ : allows only \

\/ : allows /

* : allows *

etc...

The usage of Regular Expressions in JavaScript

To use regular expression for validation in JavaScript, we use match() function of string object.

Syntax: -

```
if(ele.match(regex))
{
        return true;
}
else
{
        alert("not valid");
        args.focus();
        return false;
}
```

e.g.

```
var mobile=args.value;
var regex=/^\d{10}$/;
if(mobile.match(regex))
{
        return true;
}
else
{
        alert("invalid mobile");
        args.focus();
        return false;
}
```

Example:

```
<!doctype html>
<html>
	<head>
		<title>regular Expressions</title>
		<script type="text/javascript">
		function validateEmail(args)
		{
			var email=args.value;
			var regex=/^\w+\@\w+\.[a-zA-Z]{2,4}$/;
			if(email.match(regex))
			{
				return true;
			}
			else
			{
				alert("invalid email");
				args.focus();
				return false;
			}
		}

		function validateForm()
		{
			var email=document.getElementById("txtemail");
			if(validateEmail(email))
			{
				return true;
			}
			else
			{
```

```
                    return false;
                }
            }
            </script>
        </head>
        <body>
            <form name="form1" onsubmit="return validateForm()">
            E-Mail : <input type="text" id="txtemail" /> <br/>
            <input type="submit" value="submit"/>
            </form>
        </body>
</html>
```