JavaScript

CHAPTER V

reg[ular]
expr[essio]n

Ths. Hồ Đức Lĩnh
Mail: duclinh47th@gmail.com
Phone: 0905 28 68 87

Regular Expressions

- are the syntax you use to match and manipulate strings Or
- are patterns used to match character combinations in strings;
- a way to describe patterns in string data;
- In JavaScript, regular expressions are used with the ***RegExp*** object and some syntax called *regular expression literals*.
- These elements provide a **powerful** way to work with strings of **text** or **alphanumerics**

Regular Expressions

- Here's a regular expression to match the word *JavaScript*:

var myRegex = /JavaScript/;

→The regular expression **shown** would **match** the string "**JavaScript**" **anywhere** that it **appeared** within **another string**.

- →would match in the sentence "**This is a book about JavaScript**,"
- →and it would match in the string "**ThisIsAJavaScriptBook**,"
- →but it would not match "**This is a book about javascript**," because **regular expressions** are **case sensitive**

The syntax of regular expressions

Character	Description
^	Sets an anchor to the beginning of the input.
$	Sets an anchor to the end of the input.
.	Matches any character.
*	Matches the previous character zero or more times. Think of this as a wildcard.
+	Matches the previous character one or more times.
?	Matches the previous character zero or one time.
()	Places any matching characters inside the parentheses into a group. This group can then be referenced later, such as in a replace operation.
{n, }	Matches the previous character at least *n* times.
{n,m}	Matches the previous character at least *n* but no more than *m* times.
[ ]	Defines a character class to match any of the characters contained in the brackets. This character can use a range like 0–9 to match any number or like a–z to match any letter.
[^]	The use of a caret within a character class negates that character class, meaning that the characters in that class cannot appear in the match.
\	Typically used as an escape character, and meaning that whatever follows the backslash is treated as a literal character instead of as having its special meaning. Can also be used to define special character sets, which are shown in Table 4-7.

Common character sequences in JavaScript regular expressions

Character	Match
\b	Word boundary.
\B	Nonword boundary.
\c	Control character when used in conjunction with another character. For example, \cA is the escape sequence for Control-A.
\d	Digit.

Character	Match
\D	Nondigit.
\n	Newline.
\r	Carriage return.
\s	Single whitespace character such as a space or tab.
\S	Single nonwhitespace character.
\t	Tab.
\w	Any alphanumeric character, whether number or letter.
\W	Any nonalphanumeric character.

Creating a regular expression

- You construct a regular expression in one of two ways:
  - Using a regular expression literal, which consists of a pattern enclosed between slashes, as follows:

```
var re = /ab+c/;
```

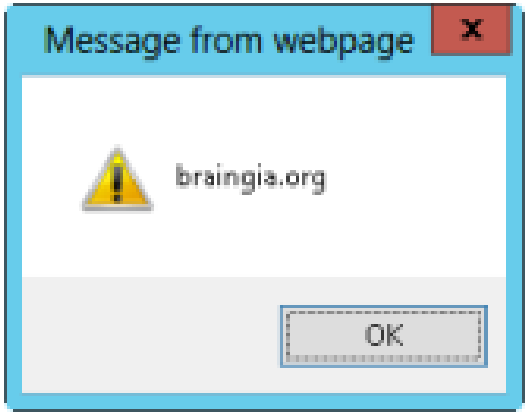
  - Or calling the constructor function of the RegExp object, as follows:

```
var re = new RegExp('ab+c');
```

Working with regular expressions

- Regular expressions are used with the **RegExp** methods test and exec
- and with the **String** methods **match**, **replace**, **search**, and **split**

```
var myString = "http://www.braingia.org";
var myRegex = /http:\/\/\w+\.(.*)/i;
var results = myRegex.exec(myString);
alert(results[1]);
```



Modifier	Description
g	Perform a global match (find all matches rather than stopping after the first match)
i	Perform case-insensitive matching
m	Perform multiline matching

- The regular expression contains several important elements.

```
var myString = "http://www.braingia.org";
var myRegex = /http:\/\/\w+\.(.*)/i;
var results = myRegex.exec(myString);
alert(results[1]);
```

- It begins by looking for the literal string *http:*
- The two forward slashes follow, but because forward slashes (/) are special characters in regular expressions, you **must escape them by using backslashes (\)**,making the regular expression *http:\/\/* to this point.
- The next part of the regular expression, ***\w***, looks for any single **alphanumeric character** → **adds** a special character ***+*** to indicate that the regular expression must **find** an **alphanumeric character at least once** and possibly **more than** once;

```
var myString = "http://www.braingia.org";
var myRegex = /http:\/\/\w+\.(.*)/i;
var results = myRegex.exec(myString);
alert(results[1]);
```

- You need to account for the dot character between the host name (*www*) and the domain name (*braingia.org*).
- You accomplish this by adding a dot character (.), but because the dot is also a special character, you need to escape it with \.. You now have ***http:\/\/\w+\.,*** which matches all the elements of a typical address right up to the domain name

```
var myString = "http://www.braingia.org";
var myRegex = /http:\/\/\w+\.(.*)/i;
var results = myRegex.exec(myString);
alert(results[1]);
```

- Finally, you need to capture the domain and use it later, so place the domain inside parentheses ( ).
- Because you don't care what the domain is or what follows it, you can use two special characters:
  - **The dot, to match any character;**
  - **and the asterisk, to match any and all of the previous characters, which is any character**

Looping over matches

- scan through all occurrences of a pattern in a string, in a way that gives us access to the match object in the loop body

```
var input = "A string with 3 numbers in it... 42 and 88.";
var number = /\b\d+\b/g;
var match;
while (match = number.exec(input)) {
  console.log("Found", match[0], "at", match.index);
}
```

```
Found 3 at 14
Found 42 at 33
Found 88 at 40
```

Methods that use regular expressions

Methods that use regular expressions

Method	Description
`exec`	A `RegExp` method that executes a search for a match in a string. It returns an array of information or null on a mismatch.
`test`	A `RegExp` method that tests for a match in a string. It returns true or false.
`match`	A `String` method that executes a search for a match in a string. It returns an array of information or null on a mismatch.
`matchAll`	A `String` method that returns an iterator containing all of the matches, including capturing groups.
`search`	A `String` method that tests for a match in a string. It returns the index of the match, or -1 if the search fails.
`replace`	A `String` method that executes a search for a match in a string, and replaces the matched substring with a replacement substring.
`split`	A `String` method that uses a regular expression or a fixed string to break a string into an array of substrings.

The *match* method

- The ***match*** method **returns** an **array** with the same information as the *Regexp* data type's *exec()* method:

```
var emailAddr = "suehring@braingia.com";
var myRegex = /\.com/;
var checkMatch = emailAddr.match(myRegex);
alert(checkMatch[0]); //Returns .com
```

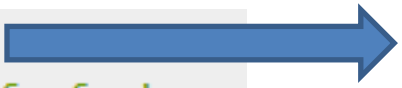
- This can be used in a conditional to determine whether a given email address contains the string *.com*:

```
var emailAddr = "suehring@braingia.com";
var myRegex = /\.com/;
var checkMatch = emailAddr.match(myRegex);
if (checkMatch !== null) {
    alert(checkMatch[0]); //Returns .com
}
```

Example

`re = /\w+\s/g` creates a regular expression that looks for one or more characters followed by a space, and it looks for this combination throughout the string.

```
1  var re = /\w+\s/g;
2  var str = 'fee fi fo fum';
3  var myArray = str.match(re);
4  console.log(myArray);
```



```
var re = new RegExp('\\w+\\s', 'g');
```

```
["fee ", "fi ", "fo "]
```

\s	Single whitespace character such as a space or tab.
\S	Single nonwhitespace character.
\t	Tab.
\w	Any alphanumeric character, whether number or letter.
\W	Any nonalphanumeric character.

The *search* method

- The *search* method works in much the same way as the *match* method but **sends back only the index (position) of the frist match**

```
var emailAddr = "suehring@braingia.com";
var myRegex = /\.com/;
var searchResult = emailAddr.search(myRegex);
alert(searchResult); //Returns 17
```

The *replace* method

- The *replace* method does just what its name implies → it replaces one string with another when a match is found
- Example: I want to **change** any **.com** email address to a **.net** email address

```
var emailAddr = "suehring@braingia.com";
var myRegex = /\.com$/;
var replaceWith = ".net";
var result = emailAddr.replace(myRegex,replaceWith);
alert(result); //Returns suehring@braingia.net
```

Regular Expression Patterns

- **Brackets** are used to find a range of characters:

Expression	Description
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x|y)	Find any of the alternatives separated with |

```
console.log(/[0123456789]/.test("in 1992"));
// → true
console.log(/[0-9]/.test("in 1992"));
// → true
```

- **Metacharacters** are characters with a special meaning:

Metacharacter	Description
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning or at the end of a word
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

- **Quantifiers** define quantities:

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*

Example

```
var re = /(?:\d{3}|\(\d{3}\))([-\/\.])\d{3}\1\d{4}/;
```

(?: → Within non-capturing parentheses

\d{3} → Looks for three numeric characters

| → OR

\(→ a left parenthesis

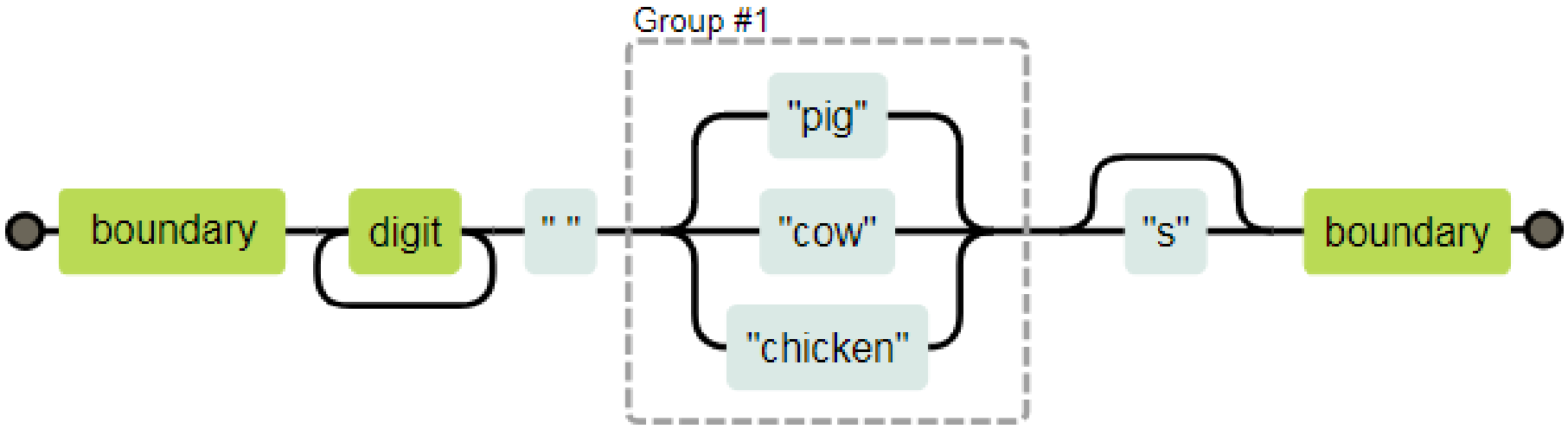
\) → a close parenthesis or end non-capturing parenthesis

([-\/\.]) → one dash, forward slash, or decimal point and when found, remember the character

\1 → decimal point (.)

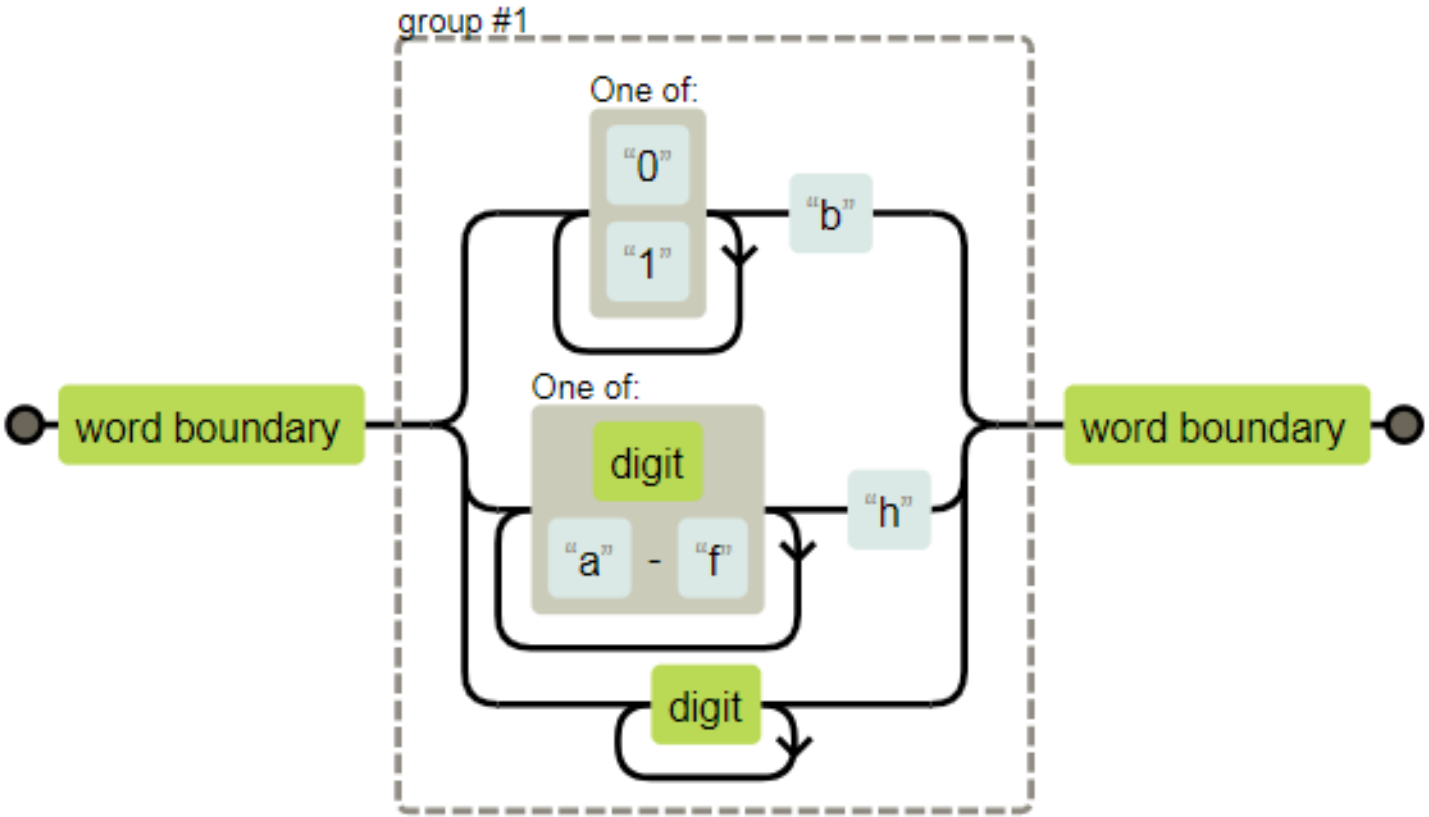
###-###-####.

The mechanics of matching



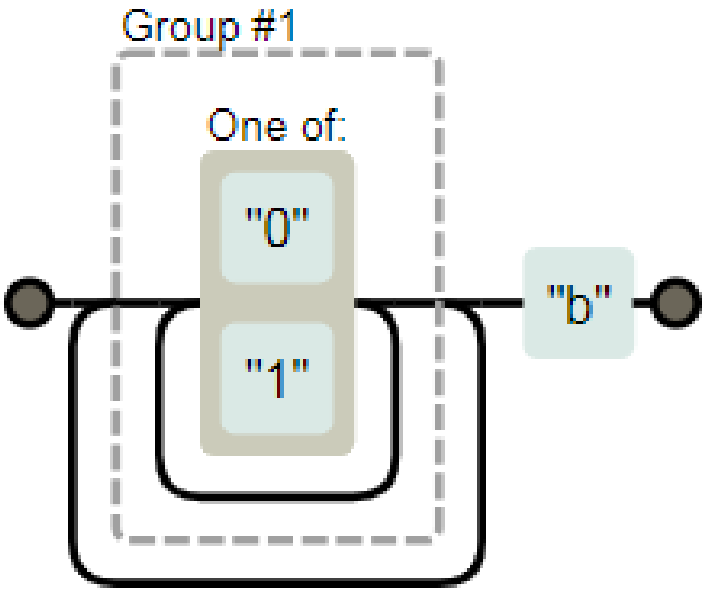
Backtracking

- The regular expression: **/\b([01]+b|[\da-f]+h|\d+)\b/**

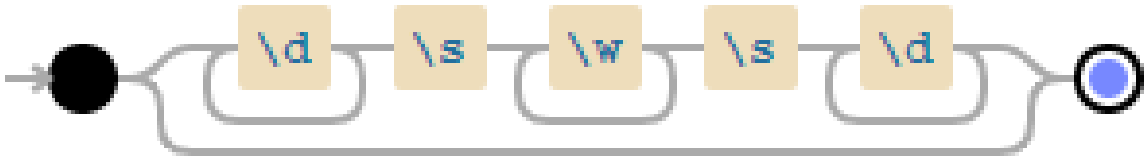


Example

- The regular expression: **/([01]+)+b/**



Example




```
1   \d+\s\w+\s\d+
```

Result: **Matches** starting at the black triangle slider

```
1   1212 character 233
```

Summary

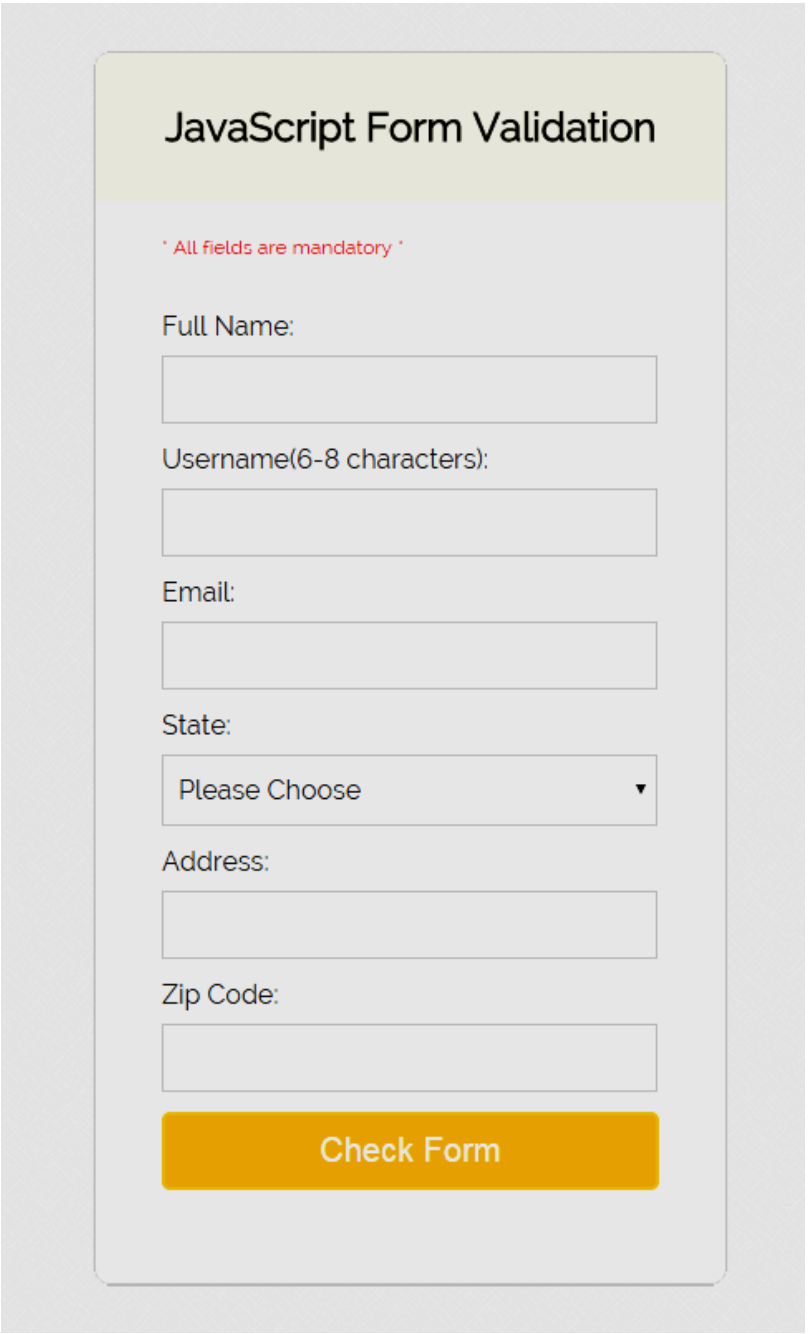
`/abc/`	A sequence of characters
`/[abc]/`	Any character from a set of characters
`/[^abc]/`	Any character *not* in a set of characters
`/[0-9]/`	Any character in a range of characters
`/x+/`	One or more occurrences of the pattern `x`
`/x+?/`	One or more occurrences, nongreedy
`/x*/`	Zero or more occurrences
`/x?/`	Zero or one occurrence
`/x{2,4}/`	Two to four occurrences
`/(abc)/`	A group

`/a|b|c/`	Any one of several patterns
`/\d/`	Any digit character
`/\w/`	An alphanumeric character (“word character”)
`/\s/`	Any whitespace character
`/./`	Any character except newlines
`/\b/`	A word boundary
`/^/`	Start of input
`/$/`	End of input

https://www.debuggex.com

JavaScript

HTML Form Input Validation



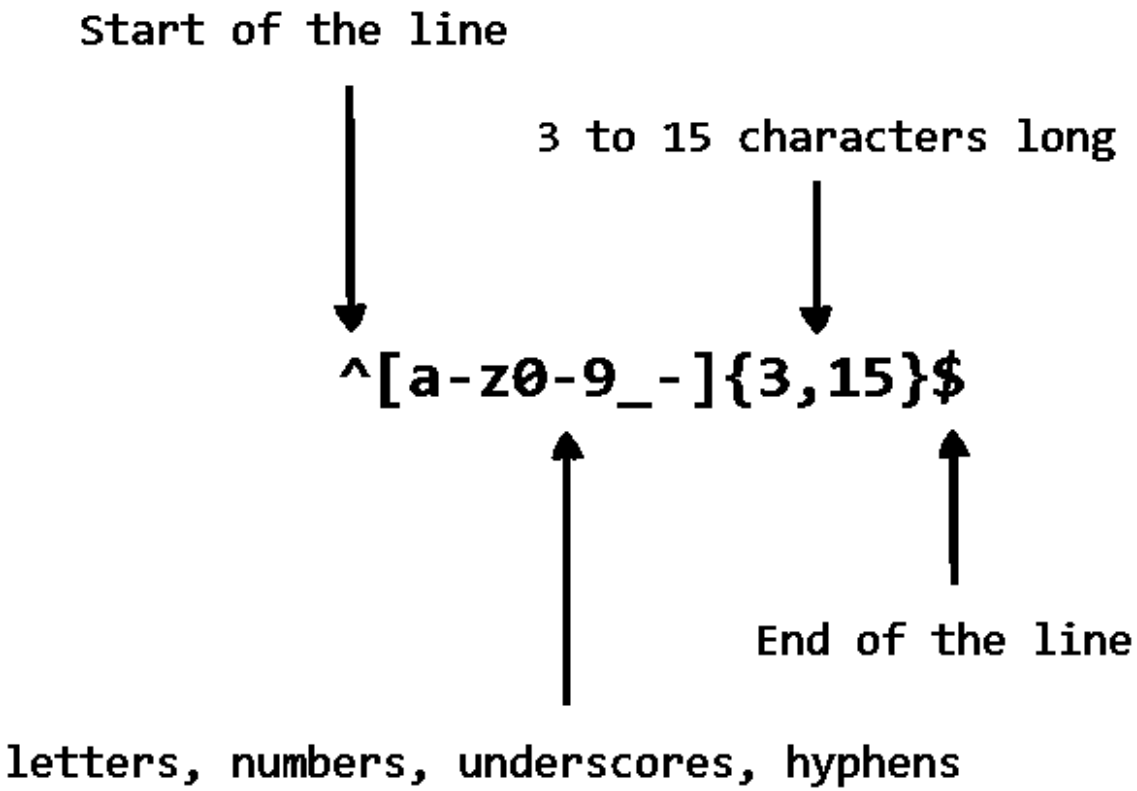
HTML FORM

What validation checks in the form?

- Check for Empty Field
- Check for Numbers
- Check for Alphabets
- Check for Numbers and Letters
- Check for Characters
- Select for Drop Down List Item
- Email Validation
- Check the length
- ....

Regular Expression (Remind)

- **Example:**



Regular Expression (Remind)

Meta character	Description
.	Period matches any single character except a line break.
[ ]	Character class. Matches any character contained between the square brackets.
[^]	Negated character class. Matches any character that is not contained between the square brackets
*	Matches 0 or more repetitions of the preceding symbol.
+	Matches 1 or more repetitions of the preceding symbol.
?	Makes the preceding symbol optional.
{n,m}	Braces. Matches at least "n" but not more than "m" repetitions of the preceding symbol.
(xyz)	Character group. Matches the characters xyz in that exact order.
|	Alternation. Matches either the characters before or the characters after the symbol.
\	Escapes the next character. This allows you to match reserved characters `[ ] ( ) { } . * + ? ^ $ \ |`
^	Matches the beginning of the input.
$	Matches the end of the input.

Regular Expression (Remind)

Shorthand	Description
.	Any character except new line
\w	Matches alphanumeric characters: `[a-zA-Z0-9_]`
\W	Matches non-alphanumeric characters: `[^\w]`
\d	Matches digit: `[0-9]`
\D	Matches non-digit: `[^\d]`
\s	Matches whitespace character: `[\t\n\f\r\p{Z}]`
\S	Matches non-whitespace character: `[^\s]`

Regular Expression (Remind)

Symbol	Description
?=	Positive Lookahead
?!	Negative Lookahead
?<=	Positive Lookbehind
?<!	Negative Lookbehind

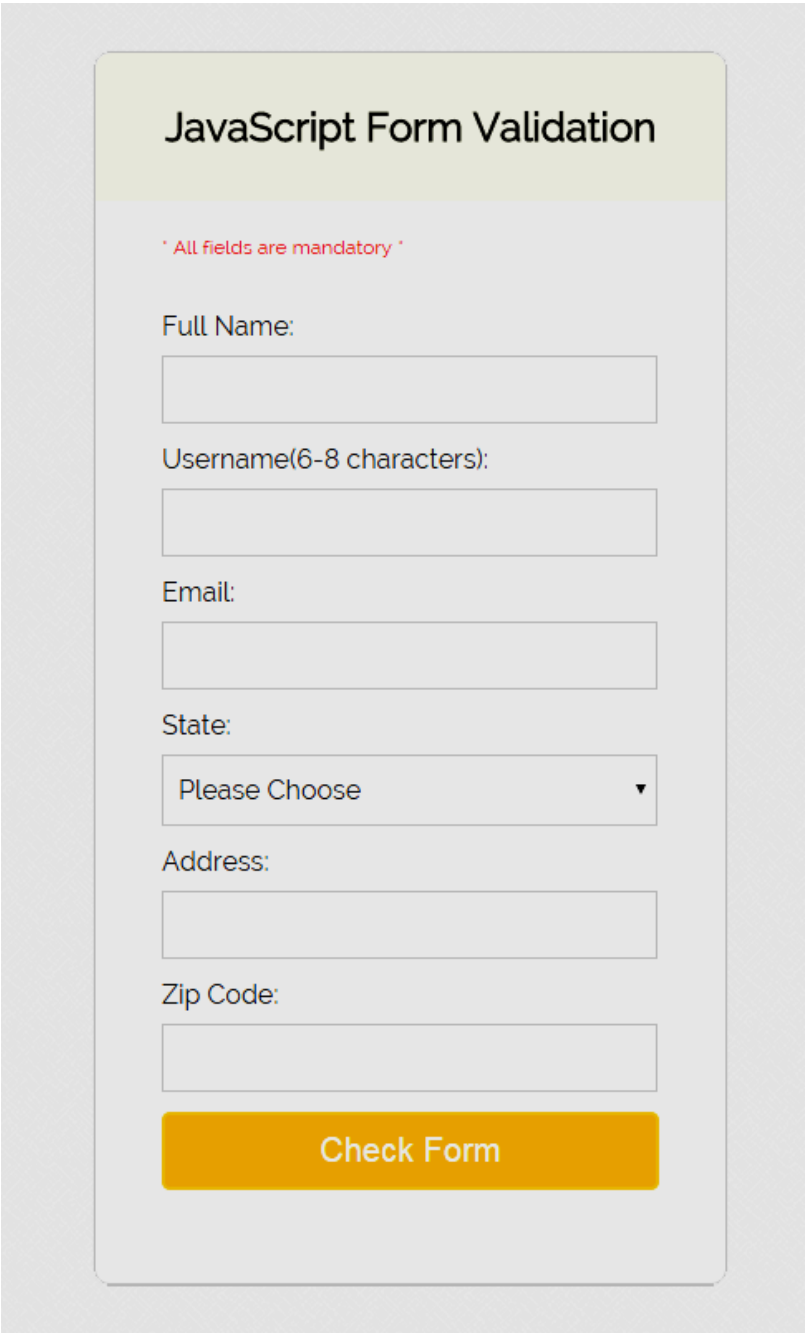
```
(T|t)he(?=\sfat) => The fat cat sat on the mat.
```

```
(T|t)he(?!\sfat) => The fat cat sat on the mat.
```

```
(?<=(T|t)he\s)(fat|mat) => The fat cat sat on the mat.
```

```
(?<!(T|t)he\s)(cat) => The cat sat on cat.
```

Flag	Description
i	Case insensitive: Sets matching to be case-insensitive.
g	Global Search: Search for a pattern throughout the input string.
m	Multiline: Anchor meta character works on each line.



HTML FORM

What validation checks in the form?

- Check for Empty Field
- Check for Numbers
- Check for Alphabets
- Check for Numbers and Letters
- Check for Characters
- Select for Drop Down List Item
- Email Validation
- Check the length
- ....

Checking for all Letters

- The expression is: **/^[a-zA-Z]+$/**

```
//Checking for all Letters
function inputAlphabet(inputElm, errElm, errMsg){
    var alphaExp = /^[a-zA-Z]+$/;
    if(inputElm.value.match(alphaExp)){
        return true;
    }else{
        errElm.innerHTML = errMsg;
        inputElm.focus();
        return false;
    }
}
```

Checking for all numbers

- The expression is: /^[0-9]+$/

```
//Checking for all numbers
function textNumeric(inputElm,errElm, errMsg){
    var numericExpression = /^[0-9]+$/;
    if(inputElm.value.match(numericExpression)){
        return true;
    }else{
        errElm.innerHTML = errMsg;
        inputElm.focus();
        return false;
    }
}
```

Checking for all numbers and letters

- The expression is: /^[0-9a-zA-Z]+$/

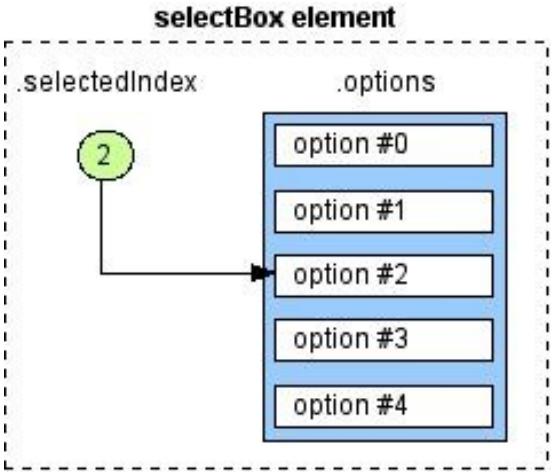
```
//Checking for all numbers and letters
function textAlphanumeric(inputElm, errElm, alertMsg){
    var alphaExp = /^[0-9a-zA-Z]+$/;
    if(inputElm.value.match(alphaExp)){
        return true;
    }else{
        //alert(alertMsg);
        errElm.innerHTML = alertMsg;
        inputElm.focus();
        return false;
    }
}
```

Restricting the length

```
//Restricting the length
function lengthDefine(inputElm, errElm, min, max){
    var uInput = inputElm.value;
    if(uInput.length >= min && uInput.length <= max){
        return true;
    }else{
        var alertMsg = "Please enter between " +min+ " and " +max+ " characters";
        //alert(alertMsg);
        errElm.innerHTML = alertMsg;
        inputElm.focus();
        return false;
    }
}
```

Right Selection made from drop-down

```
//Right Selection made from drop-down
function trueSelection(inputElm, errElm, alertMsg){
    if(inputElm.selectedIndex.value == 0){
        //alert(alertMsg);
        errElm.innerText = alertMsg;
        inputElm.focus();
        return false;
    }else{
        return true;
    }
}
```



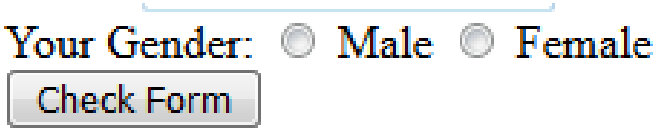
Email Validation

- The expression is: **/^[w-.+]+@[a-zA-Z0-9.-]+.[a-zA-z0-9]{2,4}$/**

```
//Email Validation
function emailValidation(inputElm, errElm, alertMsg){
    var emailExp = /^[w-.+]+@[a-zA-Z0-9.-]+.[a-zA-z0-9]{2,4}$/;
    if(inputElm.value.match(emailExp)){
        return true;
    }else{
        //alert(alertMsg);
        errElm.innerHTML = alertMsg;
        inputElm.focus();
        return false;
    }
}
```

Radio button validation

```
//Radio buttion validation
function genderValidation(inputElm, errElm, alertMsg){
    if(!inputElm.checked == false){
        return true;
    }else{
        //alert(alertMsg);
        errElm.innerHTML = alertMsg;
        return false;
    }
}
```



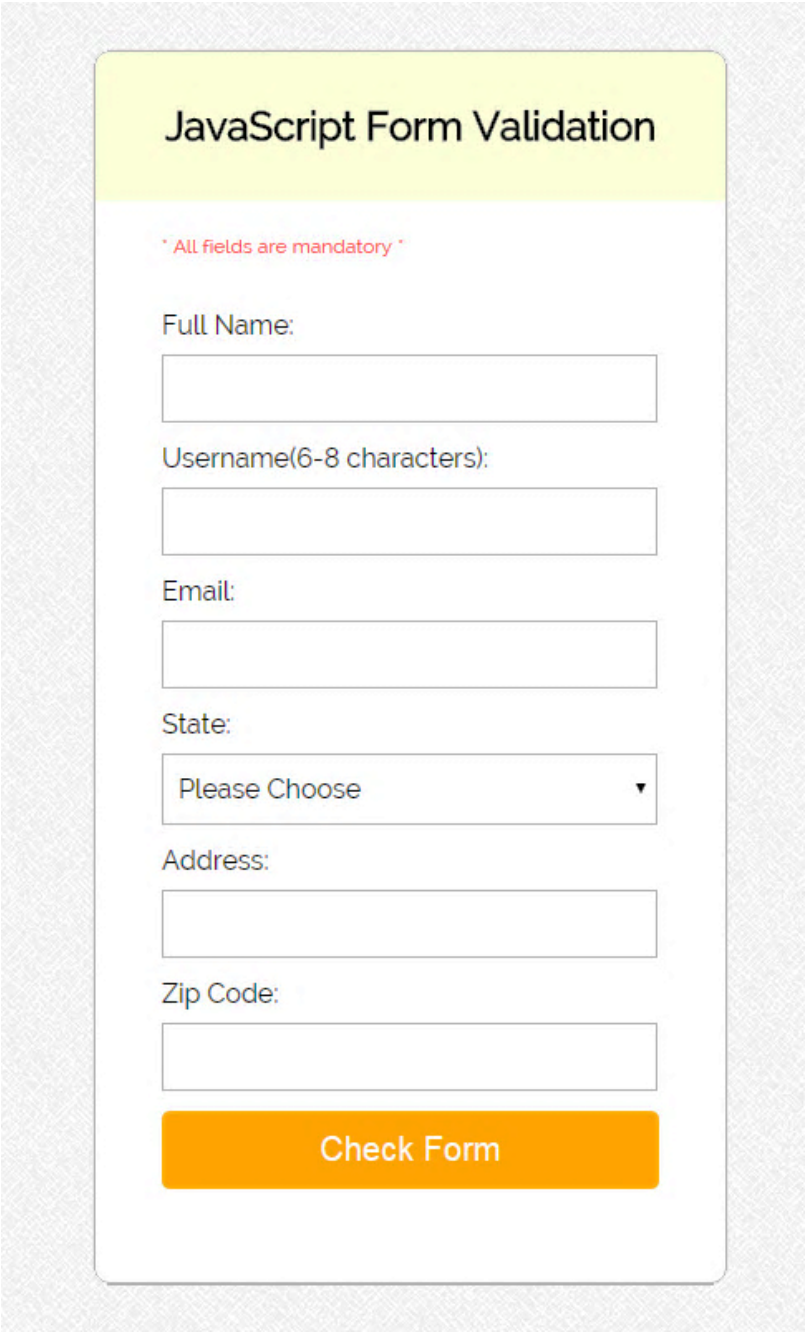
Get Form elements and call validation function

```
function formValidation(){
    //Get Element input id
    var fullname = document.getElementById('fullname');
    var phone = document.getElementById('phone');
    var gender = document.getElementById('gender');
    //Get Element show error id
    var fullname_err = document.getElementById('fullname_err');
    var phone_err = document.getElementById('phone_err');
    var gender_err = document.getElementById('gender_err');
    //Call validation function to check
    if(inputAlphabet(fullname,fullname_err,'Full name use alphabets only')){
        return true;
    }
    if(textNumeric(phone,phone_err,'Phone number use only number, please')){
            return true;
    }
    if(genderValidation(gender,gender_err,'Ban chua chon Gioi tinh')){
            return true;
    }
    return false;
}
```

HTML Form Example

```
<body>
    <h2>JavaScript Form Validation</h2>
    <form id="form_test" onsubmit="return formValidation()">
        <label>Full name:</label>
        <input type="text" name="fullname" id="fullname">
        <span id="fullname_err"></span>
        <br>
        <label>Phone:</label>
        <input type="text" name="phone" id="phone">
        <span id="phone_err"></span>
        <br>
        Your Gender:
        <input type="radio" name="gender" value="Male" id="gender"> Male
        <input type="radio" name="gender" value="Female" id="gender"> Female
        <span id="gender_err"></span>
        <br>
        <input id="submit" type='submit' value='Check Form'>
    </form>
</body>
```

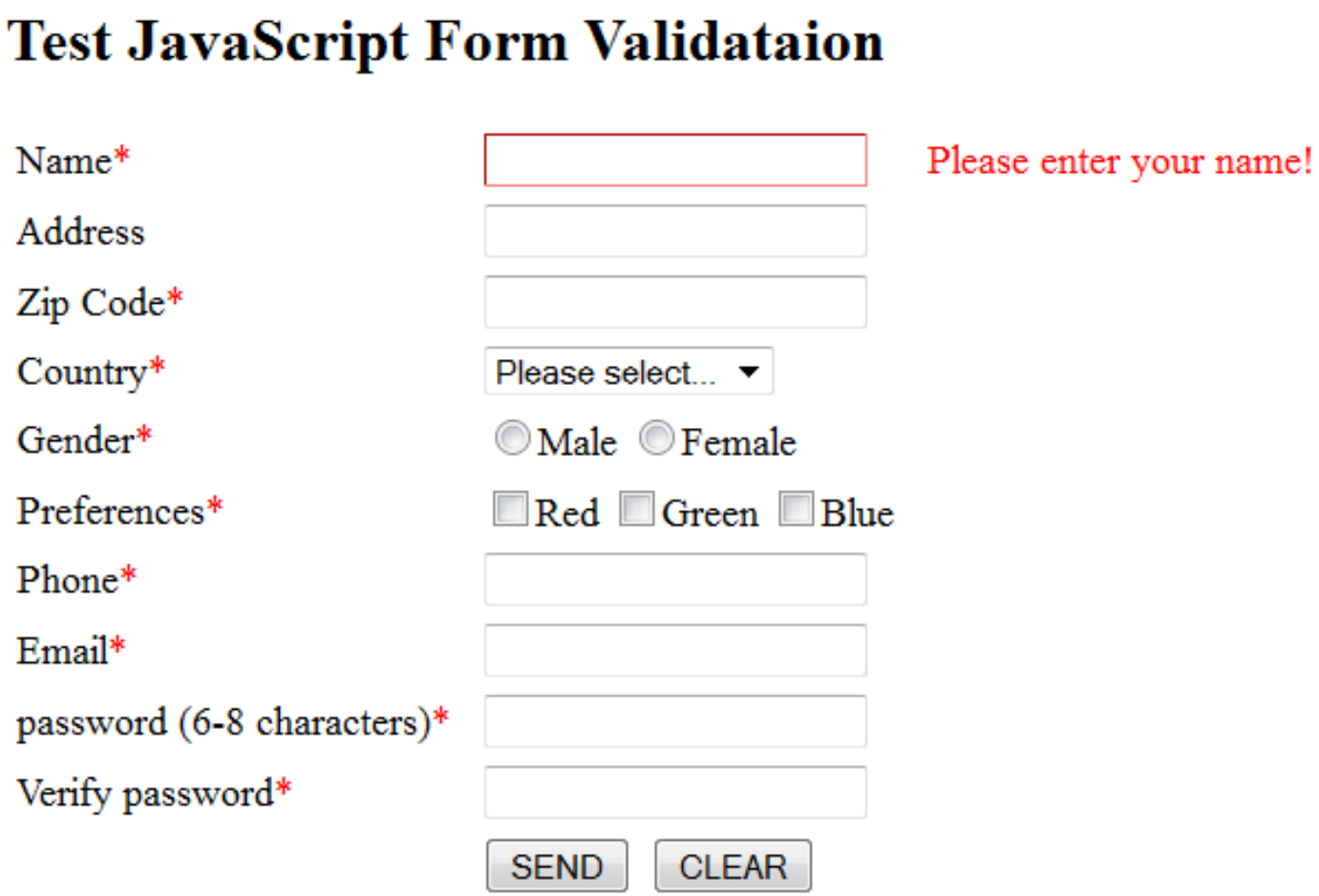
```
#fullname_err, #phone_err, #gender_err{
    display: inline-block;
    color: red;
}
```



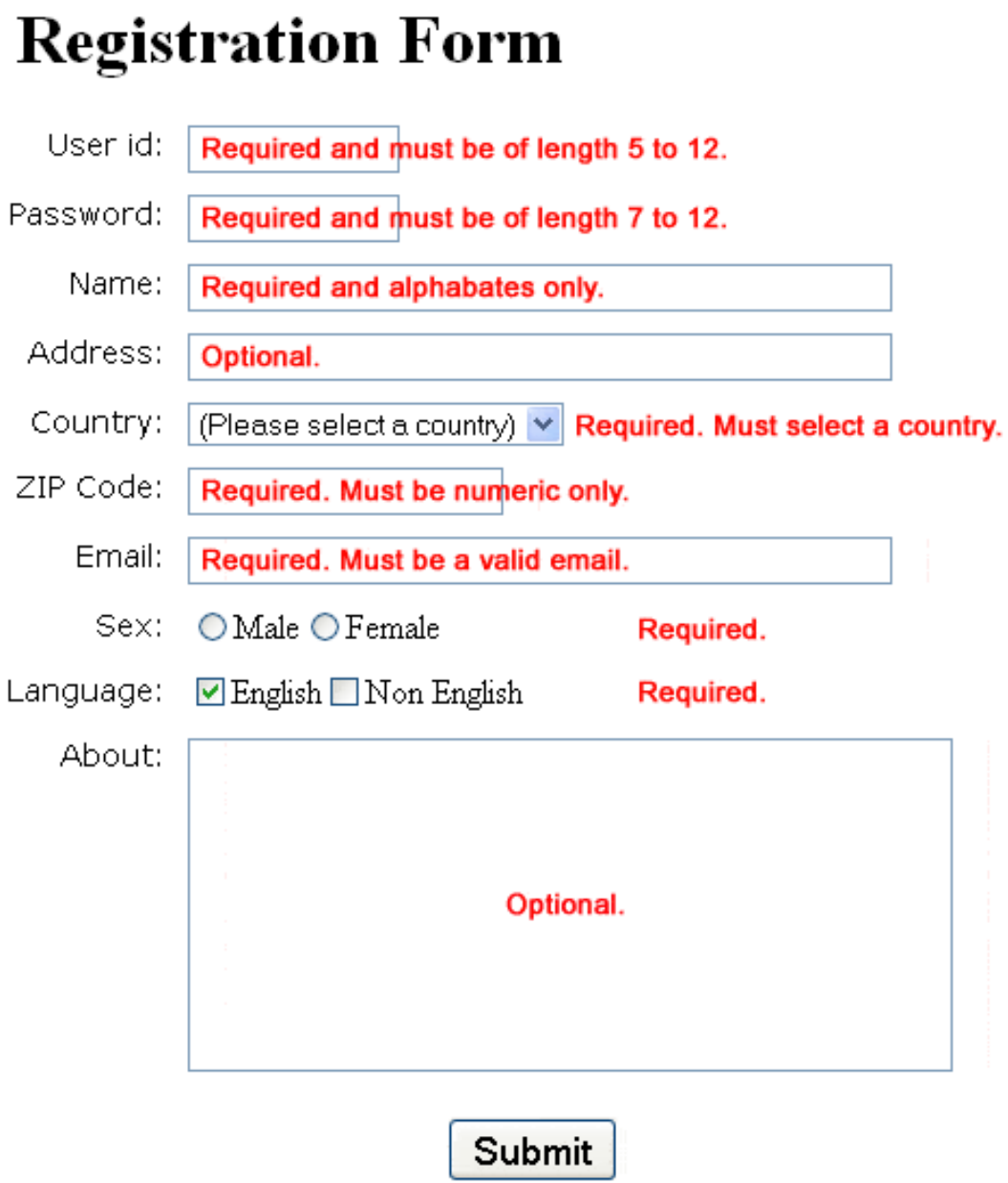
Completing other validation checks

- **Check for Empty Field**
- **Check for Numbers**
- **Check for Alphabets**
- **Check for Numbers and Letters**
- **Check for Characters**
- **Select for Drop Down List Item**
- **Email Validation**
- **Check the length**
- **….**

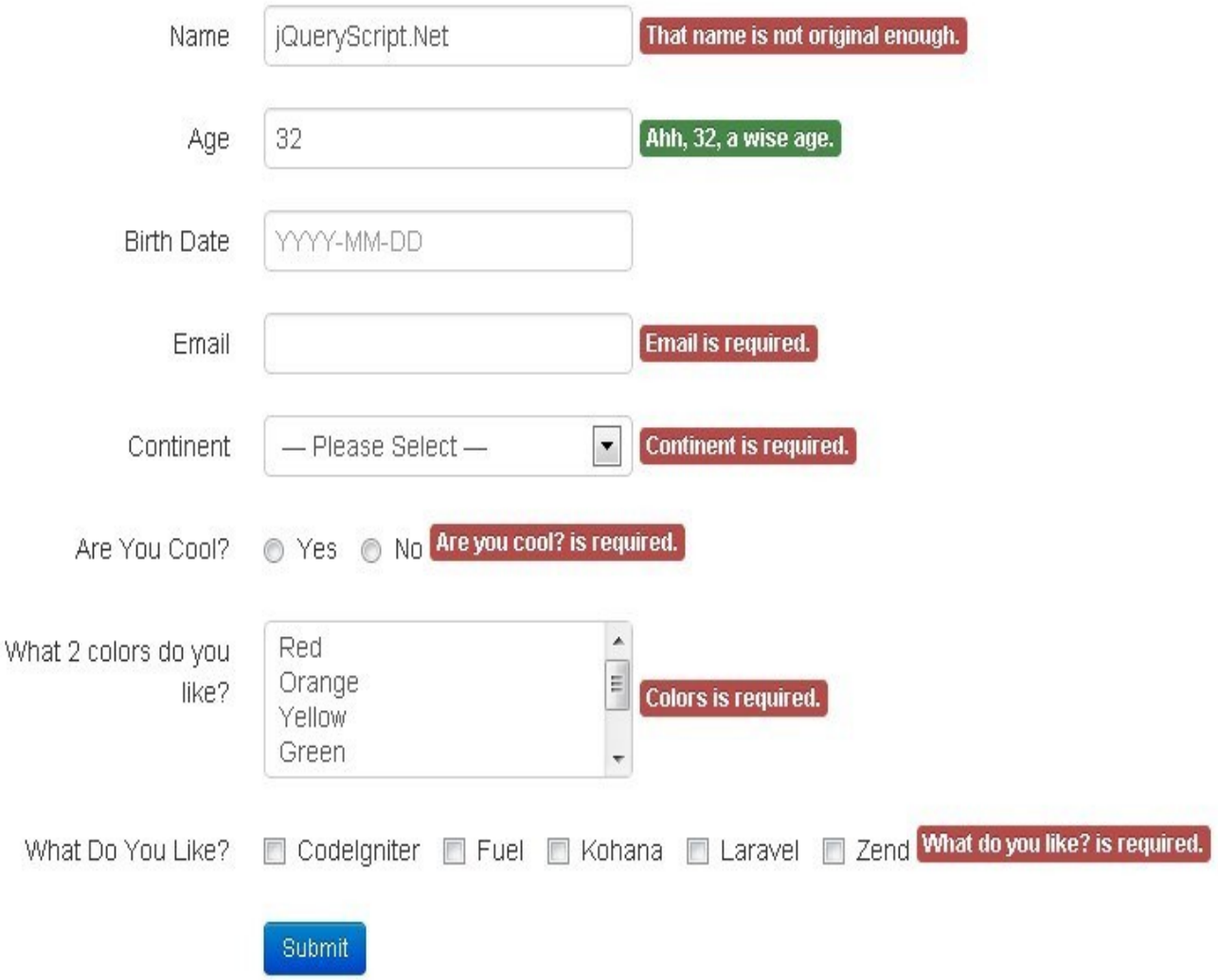
Practices 1


Test JavaScript Form Validataion
Name*
Please enter your name!
Address
Zip Code*
Country*
Please select...
Gender*
Male
Female
Preferences*
Red
Green
Blue
Phone*
Email*
password (6-8 characters)*
Verify password*
SEND
CLEAR

Practices 2


Registration Form
User id:
Required and must be of length 5 to 12.
Password:
Required and must be of length 7 to 12.
Name:
Required and alphabates only.
Address:
Optional.
Country:
(Please select a country)
Required. Must select a country.
ZIP Code:
Required. Must be numeric only.
Email:
Required. Must be a valid email.
Sex:
Male
Female
Required.
Language:
English
Non English
Required.
About:
Optional.
Submit

Practices 3


Name
jQueryScript.Net
That name is not original enough.
Age
32
Ahh, 32, a wise age.
Birth Date
YYYY-MM-DD
Email
Email is required.
Continent
— Please Select —
Continent is required.
Are You Cool?
Yes
No
Are you cool? is required.
What 2 colors do you like?
Red
Orange
Yellow
Green
Colors is required.
What Do You Like?
CodeIgniter
Fuel
Kohana
Laravel
Zend
What do you like? is required.
Submit