# Web Application Development

JavaScript
part 2

## Contents

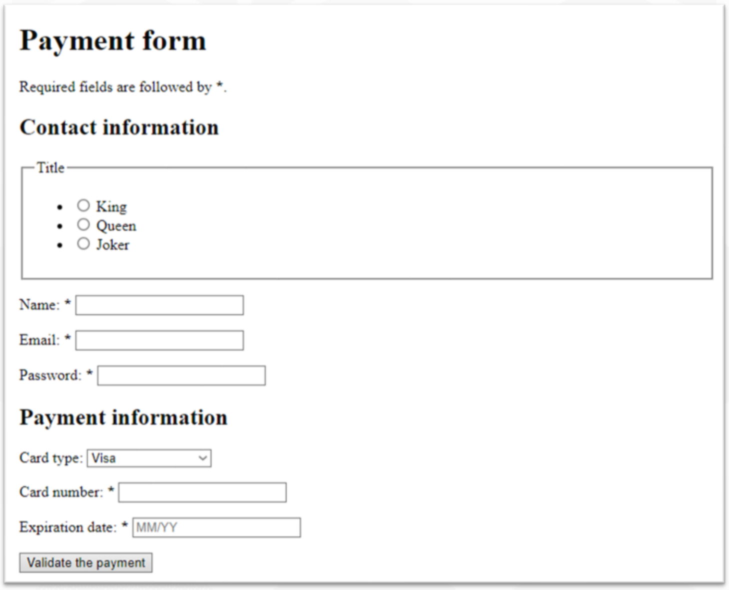
- ❑ HTML Forms
- ❑ Regular Expressions
- ❑ JavaScript Validation
- ❑ JQuery

## Contents

## What are web forms?

- ❑ Web forms are one of the main points of interaction between a user and a website or application. Forms allow users to enter data, which is generally sent to a web server for processing and storage or used on the client-side to immediately update the interface in some way.
- ❑ A web form's HTML is made up of one or more form controls, plus some additional elements to help structure the overall form — they are often referred to as HTML forms.

## Example



Payment form

Required fields are followed by *.

Contact information

Title

- ◯ King
- ◯ Queen
- ◯ Joker

Name: *

Email: *

Password: *

Payment information

Card type: Visa

Card number: *

Expiration date: * MM/YY

## The <form> Element

❑ The HTML <form> element is used to create an HTML form for user input.

```
<form name="…" action="…" method="…">
      <!- form elements-->
</form>
```

❑ Attributes of <form>

- ❑ NAME : Name of the form.
- ❑ ACTION : Specifies the webpage that will process the data from this form when the SUBMIT button is clicked.
- ❑ METHOD : Determines the method to transfer data (POST, GET)

## Example

login.html

```
<html>
   <body>
      <form  name="frmlogin"
             action="/admin/login"
             method="Post">
             ……………….
      </form>
   </body>
</html>
```

## Form Controls

- ❑ Text field
- ❑ Password field
- ❑ Hidden Text field
- ❑ Check box
- ❑ Radio button
- ❑ File Form Control
- ❑ Submit Button, Reset Button, Generalized Button
- ❑ Multiple-line text field
- ❑ Label
- ❑ Pull-down menu
- ❑ Scrolled list
- ❑ Field Set

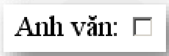
## The <label> Element

- ❑ The <label> element defines a label for several form elements.
- ❑ Syntax

```
<label
    for = IDString
    class=string
    style=string
>
```

- ❑ The for attribute of the <label> tag should be equal to the id attribute of the <input> element to bind them together
- ❑ Ex. :

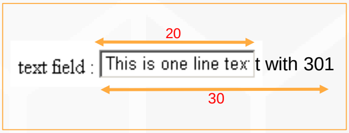
```
<label for="Languages">Anh văn: </label>
<input type="checkbox" name="Languages" id="Languages" value="Eng">
```



## Text Field

- ❑ The <input type="text"> defines a single-line input field for text input
- ❑ Syntax

```
<input
    type          = "text"
    name          = string
    readonly
    size          = variant
    maxlength     = long
    tabindex      = integer
    value         = string
    …………
>
```



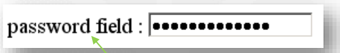
- ❑ Ex. :

```
<input type="text" name="txtName" value="This is one line text with
   301" size="20" maxlength="30">
```

## Password Field

- ❑ The <input type="password"> defines a password field (characters are masked)
- ❑ Ex. :

```
<input type="password" name="txtPassword" value="123456abc1234"
    size="20" maxlength="30">
```



*Always add the <label> tag for best accessibility practices!*

## Hidden Text Field

- ❑ The <input type="hidden"> defines a hidden input field.
- ❑ A hidden field lets web developers include data that cannot be seen or modified by users when a form is submitted
- ❑ Syntax:

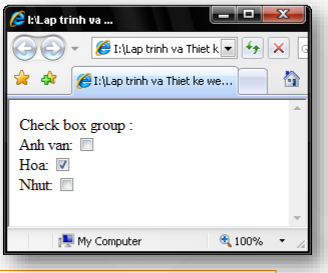
```
<input
   type          = "hidden"
   name          = string
   readonly
   size          = variant
   maxlength     = long
   tabindex      = integer
   value         = string
   ............
>
```

- ❑ Ex. : hidden text field : <input type="hidden" name="txtHidden" value="This is hidden text.You cann't see.">

## Check box

❑ Syntax

```
<input
    type    = "checkbox"
    name    = "text"
    value   = "text"
    [checked]
>
```



Check box group :
Anh van: ☐
Hoa: ☑
Nhut: ☐

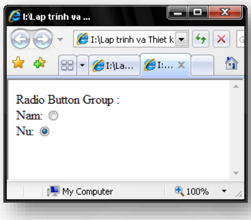
❑ Ex. :

```
<html>
    <body>
        Check box group : <br>
        Anh van: <input type="checkbox" name="Languages" value="En"><br>
        Hoa: <input type="checkbox" name="Languages" value="Chz" checked><br>
        Nhut: <input type="checkbox" name="Languages" value="Jp"><br>
    </body>
</html>
```

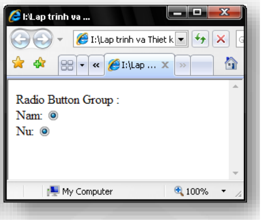
## Radio button

❑ Syntax

```
<input
    type    = "radio"
    name    = "text"
    value   = "text"
    [checked]
>
```



Radio Button Group :
Nam: ○
Nu: ●



Radio Button Group :
Nam: ●
Nu: ●

❑ Ex. :

```
<html>
    <body>
        Radio Button Group : <br>
        Nam: <input type="radio" name="sex" value="nam" checked><br>
        Nu: <input type="radio" name="sex" value="nu"checked ><br>
    </body>
</html>
```

```
<html>
    <body>
        Radio Button Group : <br>
        Nam: <input type="radio" name="sex1" value="nam" checked><br>
        Nu: <input type="radio" name="sex2" value="nu"checked ><br>
    </body>
</html>
```

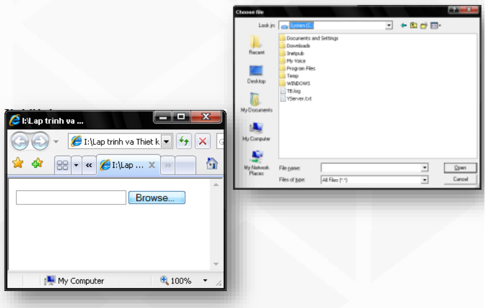
## File upload Control

- The <input type="file"> defines a file-select field and a "Browse" button for file uploads.
- Syntax

```
<form action=“…” method=“post” enctype=“multipart/form-data”
name=“...”>
    <input type=“file” name=“…” [multiple]>
</form>
```

- Ex. :

```
<html>
<body>
    <form name=“frmMain” action=“POST”
enctype=“multipart/form-data”>
        <input type="file"
name="fileUpload">
    </form>
</body>
</html>
```



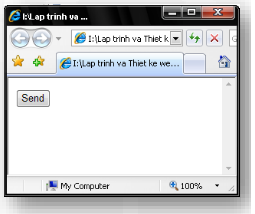
## Submit button

- <input type="submit"> defines a button for submitting form data to a form-handler. The form-handler is typically a server page with a script for processing input data
- Syntax

```
<input type=“submit” name=“…” value=“name”>
```

- Ex. :

```
<input type="submit" name="btnSend" value="Send">
```



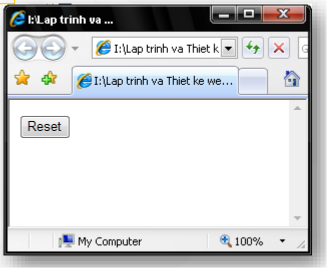
## Reset Button

- ❑ <input type="reset"> defines a reset button that will reset all form values to their default values.
- ❑ Syntax

```
<input type="reset" name="…" value="…">
```

- ❑ Ex. :

```
<input type="reset" name="btnReset" value="Reset">
```



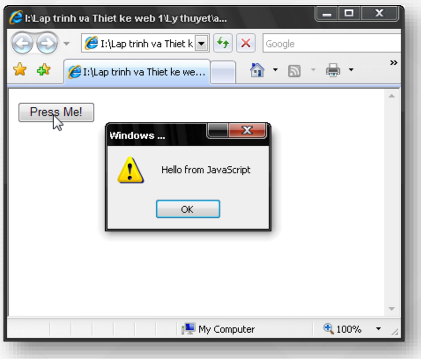
## Generalized Button

- ❑ Syntax

```
<input type="button" name="…" value="…" onclick="script">
```

- ❑ Ex. :

```
<input type="button" name="btnNormal" value="Press Me!"
onclick="alert('Hello from JavaScript');" >
```



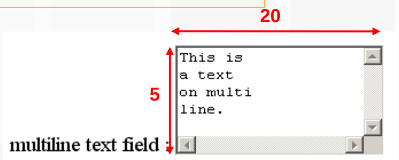
## Multi-line Text Field

- The <textarea> element defines a multi-line input field (a text area)
- Syntax

```
<textarea
    cols            = long
    rows            = long
    disabled
    name            = string
    readonly
    tabindex        = integer
    wrap            = off | hard | soft> ...............
</textarea>
```

- Ex. :

```
<textarea  cols="20" rows="5"
   wrap="off">
   This is a text on multiline.
</textarea>
```



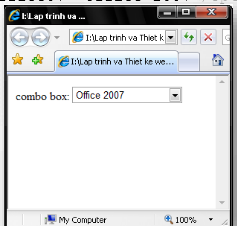
## The <select> Element

- The <select> element defines a drop-down list. The <option> element defines an option that can be selected. By default, the first item in the drop-down list is selected
- Syntax

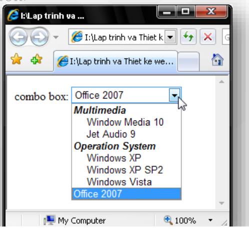
```
<select name="…">
    <optgroup label="…">
            <option [selected] value="…" >……</option>
             ………
    </optgroup>
    <option [selected] value="…" >……</option>
    ………
</select>
```

## Example

```
<html>
    <body>
          combo box:
          <select name="DSSoftware">
                    <optgroup label="Multimedia">
                              <option value="WM10">Window Media 10</option>
                              <option value="JA9">Jet Audio 9</option>
                    </optgroup>
                    <optgroup label="Operation System">
                              <option value="WXP">Windows XP</option>
                              <option value="WXPSP2">Windows XP SP2</option>
                              <option value="WVT">Windows Vista</option>
                    </optgroup>
                    <option selected value="Office07">Office 2007</option>
          </select>
    </body>
</html>
```



combo box: Office 2007

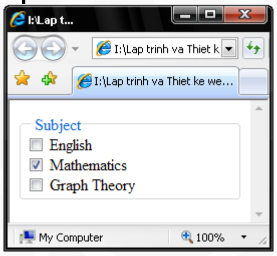


combo box: Office 2007
Multimedia
Window Media 10
Jet Audio 9
Operation System
Windows XP
Windows XP SP2
Windows Vista
Office 2007

## The <fieldset> and <legend> Elements

- The <fieldset> element is used to group related data in a form. The <legend> element defines a caption for the <fieldset> element
- Syntax

```
<fieldset>
    <legend>GroupBox's Name</legend>
    <input ……>
    …
</fieldset>
```



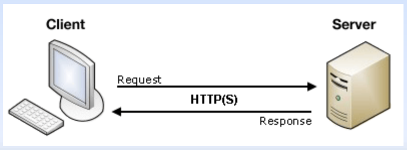
Subject
☐ English
☑ Mathematics
☐ Graph Theory

- Ex. :

```
<html>
<body>
      <fieldset>
          <legend>Subject</legend>
          <input type="checkbox" name="Subjects" value="Eng"> English<br>
          <input type="checkbox" name="Subjects" value="Math" checked> Mathematics<br>
          <input type="checkbox" name="Subjects" value="GraphTheory"> Graph Theory<br>
      </fieldset>
</body>
</html>
```

## GET/POST

**HTML Form**



## HTTP Request Methods

- ❑ The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers. The HTTP protocol provides several ways to perform a request
  - ❑ GET
  - ❑ POST
  - ❑ PUT
  - ❑ HEAD
  - ❑ DELETE
  - ❑ PATCH
  - ❑ OPTIONS
  - ❑ CONNECT
  - ❑ TRACE

## The method attribute

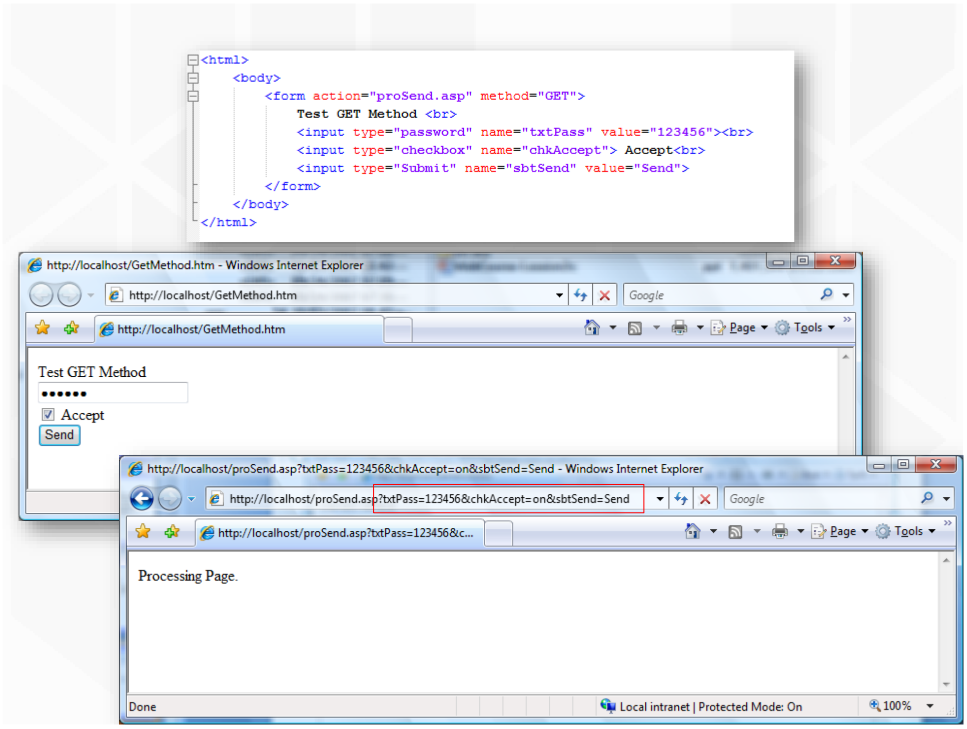
- The method attribute defines how data is sent. HTML form data can be transmitted via a number of different methods, the most common being the GET method and the POST method.
- Each time you want to reach a resource on the Web, the browser sends a request to a URL. An HTTP request consists of two parts: a header that contains a set of global metadata about the browser's capabilities, and a body that can contain information necessary for the server to process the specific request.

## The GET method

- GET is used to request data from a specified resource. Note that **the query string (name/value pairs)** is sent in the URL of a GET request.

```
/test/demo_form.php?name1=value1&name2=value2
```

- Some notes on GET requests:
  - GET requests can be cached
  - GET requests remain in the browser history
  - GET requests can be bookmarked
  - GET requests should never be used when dealing with sensitive data
  - GET requests have length restrictions
  - GET requests are only used to request data (not modify)



```
<html>
    <body>
        <form action="proSend.asp" method="GET">
            Test GET Method <br>
            <input type="password" name="txtPass" value="123456"><br>
            <input type="checkbox" name="chkAccept"> Accept<br>
            <input type="Submit" name="sbtSend" value="Send">
        </form>
    </body>
</html>
```

http://localhost/GetMethod.htm - Windows Internet Explorer

http://localhost/GetMethod.htm

Test GET Method

••••••

☑ Accept

Send

http://localhost/proSend.asp?txtPass=123456&chkAccept=on&sbtSend=Send - Windows Internet Explorer

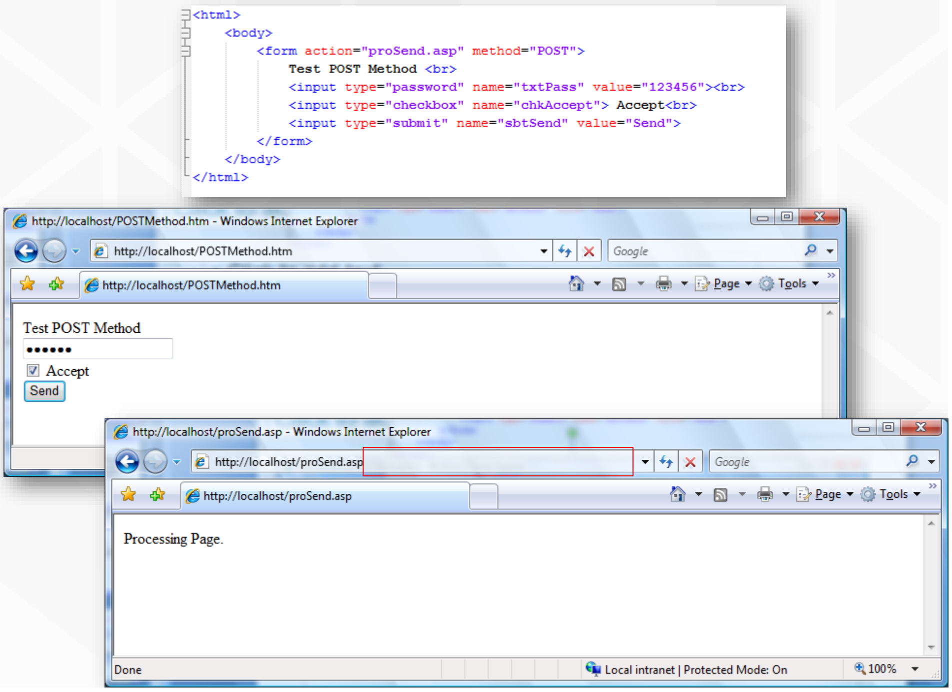
http://localhost/proSend.asp?txtPass=123456&chkAccept=on&sbtSend=Send

Processing Page.

Done | Local intranet | Protected Mode: On | 100%

## The POST Method

- ❑ POST is used to send data to a server to create/update a resource. The data sent to the server with POST is stored in the request body of the HTTP request.
- ❑ Some notes on POST requests:
    - ❑ POST requests are never cached
    - ❑ POST requests do not remain in the browser history
    - ❑ POST requests cannot be bookmarked
    - ❑ POST requests have no restrictions on data length



```
<html>
    <body>
        <form action="proSend.asp" method="POST">
            Test POST Method <br>
            <input type="password" name="txtPass" value="123456"><br>
            <input type="checkbox" name="chkAccept"> Accept<br>
            <input type="submit" name="sbtSend" value="Send">
        </form>
    </body>
</html>
```

http://localhost/POSTMethod.htm - Windows Internet Explorer

Test POST Method

••••••

☑ Accept

Send

http://localhost/proSend.asp - Windows Internet Explorer

Processing Page.

Done | Local intranet | Protected Mode: On | 100%

## Contents

- ❑ *HTML Forms*
- ❑ **Regular Expressions**
- ❑ JavaScript Validation
- ❑ JQuery

## Regular Expression

- A regular expression is a sequence of characters that forms a search pattern.
- When you search for data in a text, you can use this search pattern to describe what you are searching for.
- A regular expression can be a single character, or a more complicated pattern.
- Regular expressions can be used to perform all types of text search and text replace operations.

## Syntax

- A regular expression has a general syntax with a pattern and modifiers as follows :

/pattern/flag(s)

  - **Pattern** is a sequence of characters.
  - **Flag** is a character.
- Create a regular expression object
  - “short”

```
const regexp = /patterns/flag(s);
```

  - “long”

```
const regexp = new RegExp("pattern", "flags");
```

## The structure of a regular expression

- The main components that make up the structure of a regular expression:
  - **Literal Characters**: Ordinary characters (like ‘a’, ‘1’, ‘A’, etc.) that match themselves exactly.
  - **Metacharacters**: Symbols that have special meanings (like ‘.’, ‘*’, ‘?’, etc.). Example: ‘.’ matches any character except a newline.
  - **Character Classes**: Denoted by square brackets [] and match any one of the characters within the brackets (like ‘[abc]’). There are also predefined character classes (like ‘\d’, ‘\w’, etc.).
  - **Quantifiers**: Symbols or sets of symbols that specify how often the preceding element can occur. Example: ‘+’ matches 1 or more occurrences.

## The structure of a regular expression (cont.)

- **Groups and Capturing**: Parentheses ‘()’ are used to group parts of the expression and to capture text. For example, ‘(abc)+’ matches one or more occurrences of the sequence "abc“.
- **Alternation**: The | symbol acts like a logical OR. For example, ‘a|b’ matches either “a” or “b”.
- **Backreferences**: After capturing, you can use the captured data later in the regex. This is done using ‘\1’, ‘\2’, etc., where the number refers to the capture group
- **Escape Sequences**: The backslash ‘\’ is used to escape any metacharacter, making it a literal. For example, to match a period (.) you'd use ‘\.’ in your regex

## Example

```
const str = “Web Application Development";
const regex = /o/;
console.log(str.match(regex));
```

```
[
   "o",
   index: 13,
   input: "Web Application Development",
   groups: undefined
]
```

## Using Meta Characters

```
const str = "Web Application Development";
const regex = /.e./;
console.log(str.match(regex));
```

```
[
   “Web",
   index: 0,
   input: "Web Application Development",
   groups: undefined
]
```

## Metacharacters

| Metacharacter | Description |
|---|---|
| . | Search single characters, except line terminator or newline |
| \d | Find a digit. Equivalent to [0-9] |
| \D | Search non-digit characters i.e all the characters except digits. Equivalent to [^0-9] |
| \s | Find a whitespace character. Equivalent to [\f\n\r\t\v\u0020\u00a0\u1680\u2000\u200a\u2028\u2029\u202f\u205f\u3000\ufeff] |
| \S | Find the non-whitespace characters |
| \w | Find the word character, including the underscore. Equivalent to [A-Za-z0-9_] |
| ^ | Matches the beginning of a line or string |
| $ | Matches the end of a line or string |
| \0 | Find the NULL character |
| \t | Find the tab character |
| \n | Find the newline character |
| \uxxxx | Find the Unicode character specified by the hexadecimal number xxxx |

## Brackets & Quantifiers

| Expression | Description |
|---|---|
| (x\|y) | Find any of the alternatives between **x** or **y** separated with \| |
| [abc] | Find any of the characters inside the brackets |
| [^abc] | Find any character, not inside the brackets |
| [0-9] | Find any of the digits between the brackets |
| [^0-9] | Find any digit not in between the brackets |

| Quantifier | Description |
|---|---|
| n+ | Match any string that contains at least one **n** |
| n* | Match any string that contains zero or more occurrences of **n** |
| n? | Match any string that contains zero or one occurrence of **n** |
| m{X} | Find the match of any string that contains a sequence of **m**, **X** times |
| m{X,Y} | Find the match of any string that contains a sequence of **m**, **X** to **Y** times |
| m{X,} | Find the match of any string that contains a sequence of **m**, at least **X** times |
| ?!m | Find the match of any string which is not followed by a specific string **m** |

## Groups and backreferences

- Groups group multiple patterns as a whole, and capturing groups provide extra submatch information when using a regular expression pattern to match against a string. Backreferences refer to a previously captured group in the same regular expression.

| Characters | Meaning |
|---|---|
| (x) | Matches x and remembers the match. For example, /(foo)/ matches and remembers "foo" in "foo bar". |
| (?<Name>x) | Matches "x" and stores it on the groups property of the returned matches under the name specified by <Name>. The angle brackets (< and >) are required for group name. |
| (?:x) | Matches "x" but does not remember the match. |

## Using Modifiers

```
const str = "Web Application Development";
const regex = /[a-z]e./gi;
console.log(str.match(regex));
```

⬇

```
(3) ["Web", "Dev", "men"]
    1. 0: "Web"
    2. 1: "Dev"
    3. 2: "men"
```

## Flags

- ❑ Regular expressions have optional flags that allow for functionality like global searching and case-insensitive searching. These flags can be used separately or together in any order, and are included as part of the regular expression.
  - ❑ i: Find a character with case-insensitive matching.
  - ❑ g: Find the character globally.
  - ❑ m: Find multiline matching. Allows ^ and $ to match newline characters.
  - ❑ s: Allows . to match newline characters.
  - ❑ y: Perform a "sticky" search that matches starting at the current position in the target string.

## Example – Email validation

```
const arr = ['matuan@fit.hcmus.edu.vn',
         'matuan@gmail.com',
         'matuan.yahoo.com',
         'matuan@.com'];
const regex = /^[a-z][a-z0-9_\.]{5,32}@[a-z0-9]{2,}(\.[a-z0-9]{2,}){1,3}$/;
for (let email of arr) {
   console.log(`Check: ${email} -> ` + regex.test(email));
}
```

```
Check: matuan@fit.hcmus.edu.vn -> true
Check: matuan@gmail.com -> true
Check: matuan.yahoo.com -> false
Check: matuan@.com -> false
```

## Contents

## Data Validation

- ❑ Data Validation is the process of ensuring user input is "clean", "accurate", and "useful".
- ❑ Common types of validation include:
  - ❑ Required Fields
  - ❑ Date value
  - ❑ Numeric value
- ❑ Validation can be defined using various methods and can be implemented in many different ways:
  - ❑ Server-side validation: processed after the data is sent back.
  - ❑ Client-side validation: processed before the data is sent.

## HTML Constraint Validation

- HTML5 introduced a new HTML validation concept called **constraint validation**.
- HTML constraint validation is based on:
  - Constraint validation **HTML Input Attributes**
  - Constraint validation **CSS Pseudo Selectors**
  - Constraint validation **DOM Properties and Methods**

## Constraint Validation HTML Input Attributes

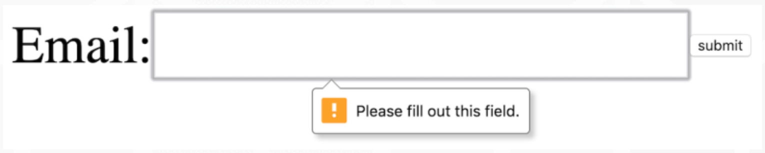
| Attribute | Description |
|---|---|
| disabled | Specifies that the input element should be disabled |
| max | Specifies the maximum value of an input element |
| min | Specifies the minimum value of an input element |
| pattern | Specifies the value pattern of an input element |
| required | Specifies that the input field requires an element |
| type | Specifies the type of an input element |

## Constraint Validation CSS Pseudo Selectors

| Selector | Description |
|---|---|
| :disabled | Selects input elements with the "disabled" attribute specified |
| :invalid | Selects input elements with invalid values |
| :optional | Selects input elements with no "required" attribute specified |
| :required | Selects input elements with the "required" attribute specified |
| :valid | Selects input elements with valid values |

## HTML Form Validation

```
<form action="#" method="GET" class="formT">
  <label for="txtEmail">Email:</label>
  <input type="text" id="txtEmail" required />
  <input type="submit" value="submit" />
</form>
```

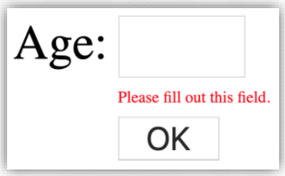


Please fill out this field.

## Constraint Validation DOM Methods

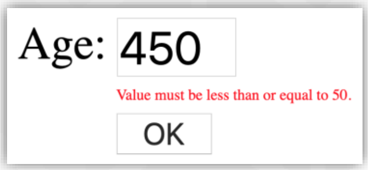
```
<label for="numAge">Age:</label>
<input type="number" id="numAge" min="10" max="50" required />
<p id="errorMsg"></p>
<input type="button" value="OK" onclick="myValid()" />
```

```
function myValid() {
   const inpObj = document.getElementById('numAge');
   if (!inpObj.checkValidity()) {
      document.getElementById('errorMsg').innerHTML = inpObj.validationMessage;
   }
}
```


Age:

Please fill out this field.

OK


Age: 450

Value must be less than or equal to 50.

OK

## Javascript Validation

```
function valid() {
   const inpAge = document.querySelector('#numAge');
   let str = inpAge.value;
   let eM = document.querySelector('#errorMsg');
   if(str.length === 0) {
      eM.innerHTML = "Can nhap tuoi";
   }
   else {
      let age = parseInt(str);
      if(age < inpAge.min) {
         eM.innerHTML = `Tuoi phai >= ${inpAge.min}`;
      }
      if(age > inpAge.max) {
         eM.innerHTML = `Tuoi phai <= ${inpAge.max}`;
      }
   }
}
```

## Contents

## JQuery

- ❑ jQuery is a fast, small, and feature-rich JavaScript library. It makes things like **HTML document traversal and manipulation**, **event handling**, **animation**, and **Ajax** much simpler with an easy-to-use API that works across a multitude of browsers.

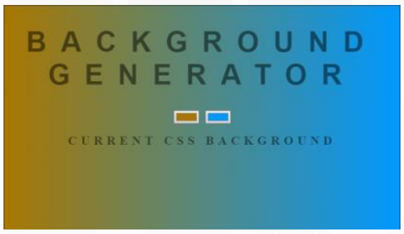
## Example

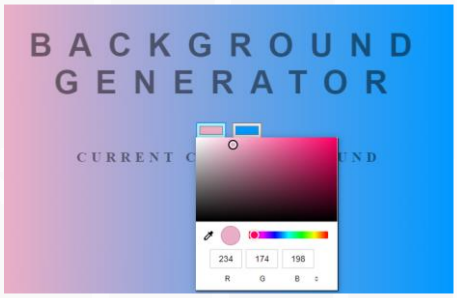
```
function jqValid() {
    if($('#numAge').val().length === 0) {
        $('#errorMsg').html('Can nhap tuoi');
    }else {
        let age = parseInt($('#numAge').val());
        if (age < $('#numAge').attr('min')) {
            $('#errorMsg').html(`Tuoi phai >= ${$('#numAge').attr('min')}`);
        }
        if (age > $('#numAge').attr('max')) {
            $('#errorMsg').html(`Tuoi phai <= ${$('#numAge').attr('max')}`);
        }
    }
}

$().ready(() => {
    $('input[type="button"]').click(jqValid);
});
```

## Exercises

❑ Create a simple website to create background color (allowing real-time color selection) as follows :





## Exercises

- ❑ Redo the week 03 exercise using: JQuery and JQuery UI.

## Exercises

- ❑ Create an HTML page for User Registration and Login with the following information :
  - ❑ **Full Name** (first letter should be capitalized).
  - ❑ **Username** (no spaces allowed, should only consist of characters, digits, and the underscore, and should not start with a digit).
  - ❑ **Email** (following the standard email format).
  - ❑ **Phone number** (following the standard 10-digit format, starting with 0).
  - ❑ **Date of birth** (in the format dd/mm/yyyy and age should be between [15, 55]).
- ❑ Requirements :
  - ❑ Layout should be centered on the screen, and labels, inputs, etc. must be aligned.
  - ❑ Upon clicking submit, the data must be validated, and the user must be clearly notified of any errors.