

SWE 363: Web Engineering & Development

Module 5-1

Client-Side Scripting (**JavaScript**)



Web Application



Objectives

- ❑ Understand the role of scripting languages in dynamic documents
- ❑ Learn how to write scripts using JavaScript
- ❑ Learn the core constructs of the JavaScript language

References

- ❑ *Internet and World Wide Web How to Program: International Edition, 5/E*, Pearson Education Inc. 2012.

- Chapter 6- JavaScript: Introduction to Scripting .
- Chapter 7: JavaScript Control Statements I
- Chapter 8: JavaScript Control Statements II
- Chapter 9: Functions
- Chapter 10: JavaScript: Arrays
- Chapter 11: Objects

- ❑ W3C- W3 Schools JavaScript Tutorial
: http://www.w3schools.com/js/js_examples.asp



- ❑ Udemy
▪ <https://www.udemy.com/>



- ❑ JSFiddle code editor
▪ <https://jsfiddle.net/>

Introduction

- ❑ Traditionally **web pages are static**, i.e. never change unless the Web page itself is changed
 - Appropriate for pages where the content and styling seldom change and where the visitor is merely a passive reader of page content.
 - Not appropriate for **dynamic pages** where layout, styling, and content need to change in response to visitor actions and desires.

- ❑ **Examples of dynamic effects**
 - Display current date and time
 - Put dynamic text into an HTML page
 - Change picture size or lightness when user clicks on accompanying buttons
 - Display block of text (e.g. revealing words definitions) when moving the mouse on top of the underlined terms being defined
 - Show contextual information in status bar
 - Validating inputs to fields before submitting a form
 - Etc.

Introduction...

- ❑ **Dynamic HTML** (or DHTML) is a collection of technologies to change static Web pages into dynamic Web pages that
 - React to events initiated by the user or by the Web page itself
 - So you can enhance page interactivity
 - Dynamically changing elements and styles
 - Generate alerts, documents, etc

- ❑ DHTML pages requires familiarity with four main topics:
 - HTML/XHTML
 - CSS
 - **JavaScript**
 - a scripting language that allows adding real programming to web pages
 - The browser's Document Object Model (DOM)
 - the collection of HTML/XHTML elements appearing on a Web page

Scripting Languages

- ❑ A scripting language is a lightweight programming language
 - is used to enhance the **functionality** and **appearance** of web pages.
 - Allow making **web pages more animated** and **more responsive** to user interaction
 - Examples: JavaScript, VBScript, Jscript, Perl, etc.
 - Interpreted **not compiled**; thus the code is executed as the page is downloaded and rendered

- ❑ The script code can be executed either
 - on the server (**server-side script**) or
 - on the client/browser (**client-side script**)

Client-Side Scripts

- ❑ **Client-Side Scripts** can perform many functions such as data validation and providing interactive feedback to the user, how:
 - Code **embedded** in Web pages along with HTML and CSS styles,
 - **downloaded** from the Web server to the browser, and
 - then **executes** locally on the client (**Interpreted by the browser**)

- ❑ **Client-side Scripting Advantages**
 - **Boost** interactivity and **reduce** the response time
 - Data **validation** before going out to the server
 - Assists **enhancing** the appearance and functionality of Web pages
 - With the DOM, it **gives more control over** all the elements on a web page

- ❑ **Client-side Scripting Disadvantages**
 - The browser **must support** the scripting language
 - Scripts may have **different results** in different browsers

What is JavaScript?

- ❑ JavaScript is one of the most simple, versatile and effective languages used to **extend functionality in websites**.
- ❑ Originally developed by **Netscape** and named **LiveScript**
- ❑ Later, **Netscape** & **Sun Microsystems** Collaboration renamed LiveScript to **JavaScript**
- ❑ Note: **Jscript** is Microsoft's implementation of JavaScript



❑ JavaScript scripting language

- Client-side scripting enhances **functionality** and **appearance**
 - Makes pages **more dynamic and interactive**
 - Pages can produce **immediate response** without contacting a server
 - **Customization** is possible on the basis of users' explicit and implicit input
 - Browser has to have a built-in (JavaScript) interpreter
- Foundation for complex server-side scripting

JavaScript is not Java

Are Java and JavaScript the Same?

- ❑ NO, JavaScript is completely different from Java except for some syntactical similarities
 - **JavaScript** - programs are interpreted in the browser
 - **Java** - programs are compiled and can be run as stand alone applications

- ❑ JavaScript **more relaxed** syntax and rules
 - fewer and "looser" data types
 - variables don't need to be declared
 - errors often silent (few exceptions)



JavaScript



JavaScript: Pros. vs Cons.

❑ Advantages:

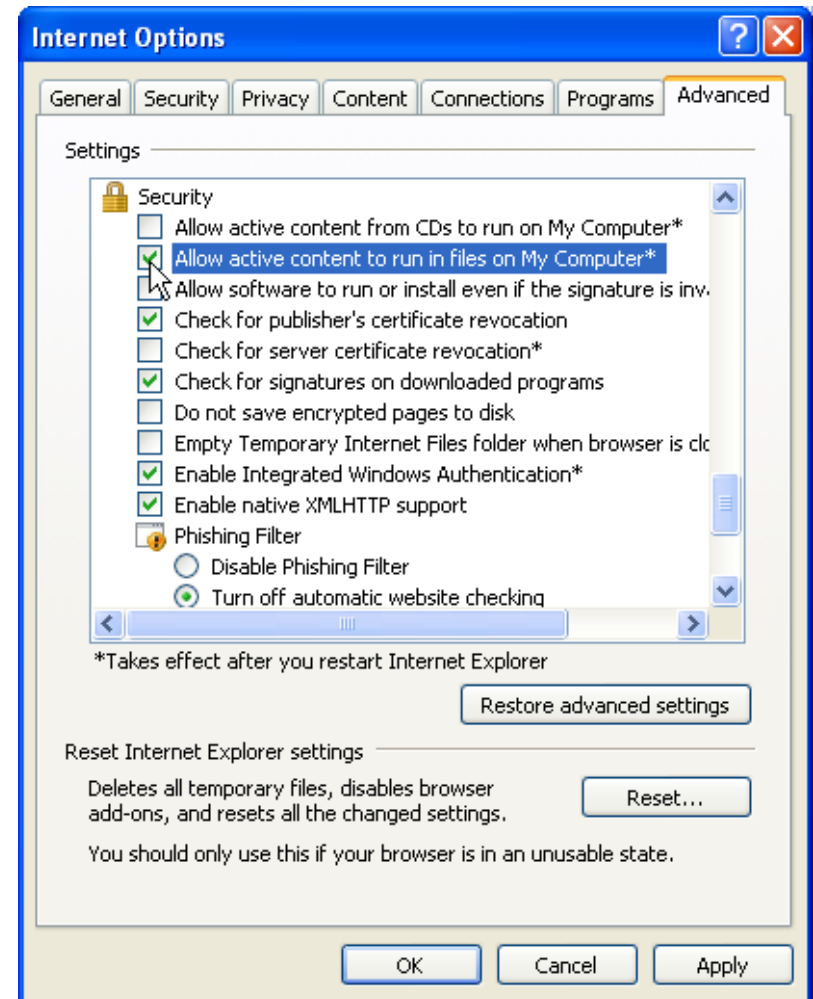
- It is **executed on the client side**- saving bandwidth and strain on the web server.
- An **easy language** - easy to learn and
- It **uses DOM model** that provides plenty of prewritten functionality
- It is **fast to the end user** - As the code is executed on the user's computer, results and processing is completed almost instantly depending on the task
- **Extended functionality** to web pages

❑ Disadvantages

- **Security Issues** - Javascript snippets, once appended onto web pages execute on client servers immediately and therefore can also be used to exploit the user's system.
- **JavaScript rendering varies**- Different layout engines may render Javascript differently resulting in inconsistency in terms of functionality and interface.

Enabling the Browser

- ❑ Before you can run code examples with JavaScript on your computer, you may need to change your browser's security settings.
- ❑ Java-enabled browser is not automatically a JavaScript-enabled browser
 - IE9 *prevents* scripts on the local computer from running by default
 - Firefox, Chrome, Opera, Safari (including on the iPhone) and the Android browser have JavaScript enabled by default.



Enabling JavaScript in Internet Explorer

Where Does JavaScript Go?

- ❑ JavaScript can be linked to an HTML page in a number of ways:
 - Inline
 - Embedded
 - External
 - External is the preferred method for cleanliness and ease of maintenance.
- ❑ Running JavaScript scripts in your browser requires (1) downloading the JavaScript code to the browser and then (2) running it.
- ❑ >> Pages with lots of scripts could potentially run slowly, resulting in a degraded experience while users wait for the page to load.
- ❑ Different browsers manage the downloading and loading of scripts in different ways, which are important things to realize when you decide how to link your scripts.

Inline JavaScript (in HTML)

- ❑ **Inline JavaScript** refers to the practice of including JavaScript code directly within certain HTML attributes

```
<body>

  <a href="#" onclick="alert('Hi')">Click Me</a>

</body>
```

```
<body>

  <input type="button" onclick="alert('Are you sure?');" />

</body>
```

- ❑ In fact, inline JavaScript is much worse than inline CSS.
- ❑ For maintenance, it requires scanning through almost every line of HTML looking for your inline JavaScript.
- ❑ >> It is a bad practice. Don't write code like this if at all possible.

Embedded JavaScript

- ❑ It refers to the practice of placing JavaScript code within a **<script> element**.

```
<body>

  <script type="text/javascript">
    document.write("<h1>Hello World!</h1>");
  </script>

</body>
```

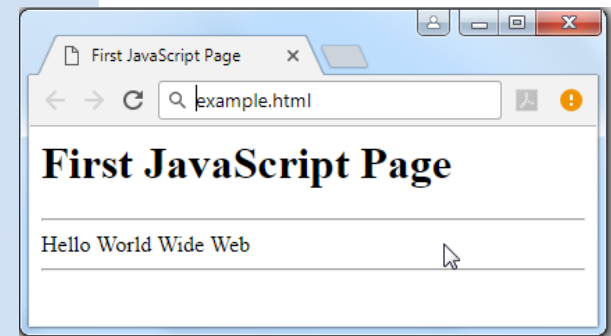
- The **type attribute** is to allow you to use other scripting languages (*JavaScript is the default*)
- This simple code does the same thing as just putting `<h1>Hello World!</h1>` in the same place in the HTML document
- The semicolon at the end of the statement is *optional (recommended)*
 - You need semicolons if you put two or more statements on the same line
- Forgetting the ending **</script> tag** for a script may prevent the browser from interpreting the script properly and may prevent the document from loading properly

Embedded JavaScript

A Simple Script

- ❑ JavaScript can be put in the `<head>` or in the `<body>` of an HTML document
 - JavaScript *functions* should be defined in the `<head>`
 - This ensures that the function is loaded before it is needed
 - JavaScript in the `<body>` will be executed as the page loads

```
<body>
<h1>First JavaScript Page</h1>
<script type="text/javascript">
    document.write("<hr>");
    document.write("Hello World Wide Web");
    document.write("<hr>");
</script>
</body>
```



>> Like with inline JavaScript, embedded scripts can be difficult to maintain.

Note: The issue is that when you run `document.write` after the document has loaded, it overwrites the entire document. If it is run before that, it does not overwrite it.

https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_write_over

External JavaScript

- ❑ JavaScript can be put in a separate **.js** file
 - `<script src="myJavaScriptFile.js"></script>`
 - Put this HTML wherever you would put the actual JavaScript code
 - An external **.js** file lets you use the same JavaScript on multiple HTML pages
 - The external **.js** file cannot itself contain a `<script>` tag
- ❑ Using **.js** file makes the code more reusable >> it can be included into any HTML5 document
- ❑ A JavaScript file has to be **loaded completely before the browser can begin any other downloads** (including images)
- ❑ For sites with multiple external JavaScript files, this can cause a **noticeable delay in initial page rendering**.

.js file Example

```
<html>
  <head><title>First JavaScript Program</title></head>
  <body>
    <h1>First JavaScript Page</h1>
    <script type="text/javascript"
      src="myJavaScriptFile.js">
    </script>
  </body>
</html>
```

myJavaScriptFile.js
document.write("<hr>");
document.write("Hello World Wide Web");
document.write("<hr>");

cannot contain a <script> tag

- ❑ Use the **src attribute** to include JavaScript codes from an external file.
- ❑ The included code is inserted in place.

JavaScript isn't always available

- ❑ Some old browsers do not recognize **script** tags
 - to tell those browsers to ignore what is in the `<script>` tag

```
<script type="text/javascript">  
  <!--  
    some JavaScript code  
  //-->  
</script>
```

- Use the `<noscript> message </noscript>` to display a message in place of whatever the JavaScript would put there

```
<noscript>  
  Your browser does not support JavaScript.  
</noscript>
```

Referencing HTML Elements

- ❑ An HTML element must be assigned an id for a script to refer to it:

```
<tag id="idValue"...>
```

- The assigned **idValue** value must be **unique** and composed of alphanumeric excluding spaces

- ❑ Once an id is assigned, the **HTML object can be referenced in a script:**

```
document.getElementById("idValue")
```

Getting and Setting Style Properties

- ❑ DHTML is created commonly by changing the style properties of HTML elements

-- Get a current style property:

```
document.getElementById("id").style.property
```

-- Set a different style property:

```
document.getElementById("id").style.property = value
```

- ❑ For example:

```
<h2 id="Head" style="color: blue"> This is a Heading</h2>
```

- We can change the color property as

```
document.getElementById("Head").style.color = "red"
```

Variables

- ❑ variables are declared with the `var` keyword (case sensitive)
- ❑ `types` are not specified, but JS does have types
 - `Number, Boolean, String, Array, Object, Function, Null, Undefined`
 - Strings are written inside double or single quotes. Numbers are written without quotes.
 - If you put a number in quotes, it will be treated as a text string.
 - You can find out a variable's type by calling `typeof`

```
var clientName = "Connie Client";  
var pi = 3.1416, x, y, name = "Dr. Dave" ;  
var weight = 127.4;  
var credits = 5 + 4 + (2 * 3);
```

- ❑ Variables declared within a function are `local` to that function (accessible only within that function)
- ❑ Variables declared outside a function are `global` (accessible from anywhere on the page)

Conditional Statements

- ❑ In JavaScript we have the following conditional statements:
 - **if statement** - use this statement if you want to execute some code only if a specified condition is true
 - **if...else statement** - use this statement if you want to execute some code if the condition is true and another code if the condition is false
 - **if...else if...else statement** - use this statement if you want to select one of many blocks of code to be executed
 - **switch statement** - use this statement if you want to select one of many blocks of code to be executed

- ❑ For examples:
 - https://www.w3schools.com/js/js_if_else.asp
 - https://www.w3schools.com/js/js_switch.asp

Because most JavaScript syntax is borrowed from C (and is therefore just like Java), we won't spend much time on it.. **But they are required**

Example: Dynamic Welcome Page

```
<script type="text/javascript">

    var txt;
    var person = prompt("Please enter your name:", "XYZ");
    if (person == null || person == "") {
        txt = "User cancelled the prompt.";
    } else {
        txt = "Hello " + person + "!";
    }
    document.write(txt);

</script>
```


JavaScript Loops

- ❑ JavaScript supports different kinds of loops:
 - **for** - loops through a block of code a number of times
 - **for/in** - loops through the properties of an object
 - **while** - loops through a block of code while a specified condition is true
 - **do/while** - also loops through a block of code while a specified condition is true

- ❑ For Examples:
 - https://www.w3schools.com/js/js_loop_for.asp
 - https://www.w3schools.com/js/js_loop_while.asp
 - https://www.w3schools.com/js/js_break.asp

JavaScript for/in Statement

- ❑ The **for/in** statement loops through the properties of an object.
- ❑ The block of code inside the loop will be executed once for each property.

```
for (var in object) {  
    code block to be executed  
}
```

<i>var</i>	Required. A variable that iterates over the properties of an object
<i>object</i>	Required. The specified object that will be iterated

```
<p id="demo"></p>  
<script>  
    var person = {fname:"John", lname:"Doe", age:25};  
    var text = "";  
    var x;  
    for (x in person) {  
        text += person[x] + " ";  
    }  
    document.getElementById("demo").innerHTML = text;  
</script>
```

John Doe 25

Statement Keywords

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

Operators

- ❑ Arithmetic operators (all numbers are floating-point):

+ - * / % ++ --

- ❑ Comparison operators:

< <= == != >= >

- ❑ Logical operators:

&& || ! (&& and || are *short-circuit* operators)

- ❑ Bitwise operators:

& | ^ ~ << >> >>>

- ❑ Assignment operators:

+= -= *= /= %= <<= >>= >>>= &= ^= |=

- ❑ String operator:

+ (concatenation)

- ❑ The conditional operator:

condition ? value_if_true : value_if_false

- ❑ Special equality tests:

- == and != try to convert their operands to the same type before performing the test
- === and !== consider their operands *unequal* if they are of different types

- ❑ Additional operators (to be discussed):

new typeof void delete

Example

```
<!DOCTYPE html>
<html> <body>

<p>Input your age and click the button:</p>
<input id="age" value="15" />

<button onclick="myFunction()">Try it</button>
<p id="demo"></p>

<script>
function myFunction() {
    var age, voteable;
    age = document.getElementById("age").value;
    voteable = (age < 18) ? "Too young":"Old enough";
    document.getElementById("demo").innerHTML = voteable +
    "to vote.";
}
</script>
</body> </html>
```

Input your age and click the button:

Too young to vote.

Input your age and click the button:

Old enough to vote.

JavaScript Functions

- ❑ A JavaScript function is a block of code designed to perform a particular task.
- ❑ A JavaScript function is executed when "something" invokes it (calls it).

```
<h2>JavaScript Functions</h2>
<p>This example calls a function to perform a calculation </p>
<p id="demo"></p>
<script>
    function myFunction(a, b) {
        return a * b;
    }
    var x= window.prompt("Enter first number: ", 0);
    var y= window.prompt("Enter Second number:", 0);

    document.getElementById("demo").innerHTML = myFunction(x, y);
</script>
```

Another Example

```
<p id="demo"> This text can be changed: </p>
<button onclick="ChangeStyle()">Try it</button>

<script type="text/javascript">
function ChangeStyle() {
    document.getElementById ("demo").style.fontSize = "14pt";
    document.getElementById("demo").style.fontWeight = "bold";
    document.getElementById("demo").style.color = "red";
}
</script>
```

- ❑ You can call the function in any statement or to handle user events, e.g.

```
<p id="demo" onclick="ChangeStyle()" >
```

More Examples

[JavaScript can change HTML content](#)

[JavaScript can change HTML attributes](#)

[JavaScript can change CSS style](#)

[JavaScript can hide HTML elements](#)

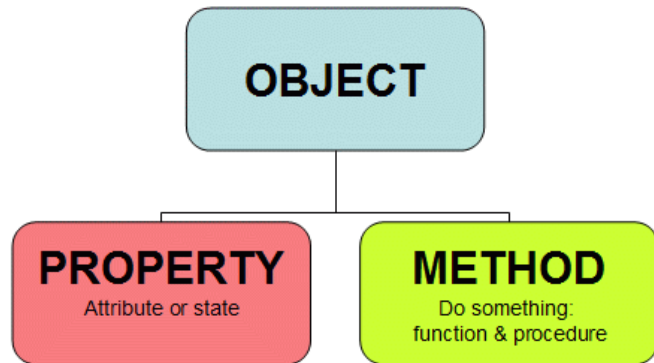
[JavaScript can show hidden HTML elements](#)

General Remarks

- ❑ Most JavaScript syntax and statements are borrowed from C and Java, so we won't spend much time on it
- ❑ JavaScript is case sensitive; all keywords are lowercase
- ❑ JavaScript uses C-style comments: `//` and `/* */`
- ❑ Every statement should end with a semicolon (although it is optional unless two or more statements appear on the same line)
- ❑ The interpreter notifies the user of a syntax error when it attempts to execute the statement containing the error (at runtime)

JavaScript Objects

Objects, Properties, and Methods



Examples of Objects: Window, Document, Form

Properties

- Variable whose value can change
- Can be assigned values for each object
- Naming Convention: uses a noun to denote attribute
- Examples `document.bgColor`
`form.name`
`window.status`

Methods

- One or more programming statements
- Function that is called from an object
- Naming Convention: uses a verb to denote action
- Examples `document.write`
`window.open`
`history.go`

Objects can also have related Events

- Actions that can trigger other functions
- Event handlers react to specific events
- Examples `onclick`
`onmouseover`
`onfocus`
`onload`

JavaScript: Object-Based Language

- ❑ JS uses many objects, but not a complete OOP language
- ❑ JS **provides** many built-in objects and it **allows** you to define and create your own
- ❑ **Native objects** are those objects supplied by JavaScript.
 - Examples of these are **String**, **Number**, **Array**, **Image**, **Date**, **Math**, etc.
- ❑ **Host objects** are objects that are supplied to JavaScript by the browser environment. Examples of these are **window**, **document**, **forms**, etc.
- ❑ **User-defined objects** are those that are defined by you, the programmer.
- ❑ **A fundamental concept** in JavaScript is that every element that can hold properties and methods is an object, except for the primitive data types.

The Math Object

- ❑ The Math object allows you to perform mathematical tasks.
- ❑ Can be accessed as *Math.function*

```
Math.round(4.7);    // returns 5
Math.round(4.4);    // returns 4

Math.pow(8, 2);     // returns 64
Math.sqrt(64);      // returns 8

Math.abs(-4.7);     // returns 4.7

Math.ceil(4.4);     // returns 5
Math.floor(4.7);    // returns 4
```

- Math.ceil(x) returns the value of x rounded **up** to its nearest integer:
- Math.floor(x) returns the value of x rounded **down** to its nearest integer

More functions: https://www.w3schools.com/jsref/jsref_obj_math.asp

Arrays

- ❑ An array is a special variable, which can **hold more than one value** at a time.

```
var name = []; // empty array
var name = [value, value, ..., value]; // pre-filled
name[index] = value; // store element
```

- ❑ You refer to an array element by referring to the *index number*.
 - **var name = colors [0];**

```
var colors = ["red", "green", "blue"];
document.getElementById("demo").innerHTML = colors[0];
```

- ❑ With JavaScript, the full array can be **accessed by referring to the array name**:

```
var colors = ["red", "green", "blue"];
document.getElementById("demo").innerHTML = colors;
```

Arrays..

- ❑ Arrays are a **special type of objects**.
- ❑ It can be defined using the **new** operator,

```
var names = new Array(5) ;
```

- ❑ Arrays in JavaScript are “**dynamic**” entities (objects) that can change size after they are created
 - JavaScript reallocates an Array when a value is assigned to an element that is outside the bounds of the original Array
 - Elements between the last element of the original Array and the new element have **undefined values**
 - Referring to an element outside the **Array** bounds is normally a **logical error**.
- ❑ Each array has a length attribute that be used to get the size of the array, e.g.

```
arrayname.length
```

Arrays..

- ❑ You can iterate through all elements of an array using **for-loop** or **for-in** loop

```
for ( var i = 0; i < theArray.length; i++ ) total1 += theArray[ i ];
```

```
for ( var element in theArray ) total2 += theArray[element];
```

- ❑ How to use the array to **show pictures randomly**?
 - store the names of the image files as strings
 - Then , access the array using a randomized index

```
var pictures = [ "CPE", "EPT", "GPP", "GUI", "PERF", "PORT", "SEO" ];
```

```
document.write ( "<img src = \"\" +  
                pictures[ Math.floor( Math.random() * 7 ) ] + ".gif\" />" );
```


Passing Arrays to Functions

- ❑ To pass the array *names* to the function `outputArray()`

`outputArray(names);`

```
<script type="text/javascript">
  var a= [3,4,5];
  document.write(a);
  document.write("<br>" + renderData(a));
  document.write("<br>" + a);
  function renderData(a){
    a.push(7);
    return a.length;
  }
</script>
```

3,4,5
4
3,4,5,7

- ❑ In the function header, specify a parameter that will receive the array, e.g.

`function outputArray(b)`

Array methods

Method	Description
<u>concat()</u>	Joins two or more arrays, and returns a copy of the joined arrays
<u>fill()</u>	Fill the elements in an array with a static value
<u>filter()</u>	Creates a new array with every element in an array that pass a test
<u>find()</u>	Returns the value of the first element in an array that pass a test
<u>findIndex()</u>	Returns the index of the first element in an array that pass a test
<u>forEach()</u>	Calls a function for each array element
<u>indexOf()</u>	Search the array for an element and returns its position
<u>join()</u>	Joins all elements of an array into a string
<u>pop()</u>	Removes the last element of an array, and returns that element
<u>push()</u>	Adds new elements to the end of an array, and returns the new length
.....	

Array methods

Example

- ❑ The `shift()` method removes the first item of an array.
 - **Tip:** To remove the last item of an array, use the [pop\(\)](#) method.

```
<html> <body>
<p>Click the button to remove the first element of the array.</p>
<button onclick="myFunction()">Try it</button>

<p id="demo"></p>
<script>
var colors = ["Red", " Blue", " Green", " White"];
document.getElementById("demo").innerHTML = colors;

function myFunction() {
    colors.shift();
    document.getElementById("demo").innerHTML = colors;
}
</script>
```

Click the button to remove the first element of the array.

Try it

Red, Blue, Green, White

Click the button to remove the first element of the array.

Try it

Blue, Green, White

Arrays and objects

- ❑ Since Arrays are a special type of objects, the *typeof* operator in JavaScript returns "object" for arrays.

- ❑ Arrays use **numbers** to access its "elements".

```
var person = ["John", "Doe", 46];
```

- e.g. **person[0]** returns John

- ❑ Objects use **names** to access its "members".

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

- e.g. **person.firstName** returns John

- ❑ In JavaScript,

- **arrays** use **numbered indexes**
- **objects** use **named indexes**.

How to Recognize an Array

- ❑ A common question is: How do I know if a variable is an array?
 - The problem is that the JavaScript operator `typeof` returns "object" because a JavaScript array is an object.

```
var colors= ["red", "blue", "green"];  
  
typeof colors;           // returns object
```

- ❑ Solution 1:
 - To solve this problem [ECMAScript5](#) defines a new method `Array.isArray()`:

```
Array.isArray(colors); // returns true
```

not supported in older browsers.

- ❑ Solution 2:
 - The `instanceof` operator returns true if an object is created by a given constructor:
`colors instanceof Array` // returns true

Remark

- ❑ There is **no need** to use the JavaScript's **built-in array** constructor **new Array()**.
 - Use **[]** instead.
- ❑ These two different statements both create a new empty array named points:

```
var points = new Array();           // Bad
var points = [];                    // Good
```

- ❑ **Do Not** Declare Strings, Numbers, and Booleans as Objects! They complicate your code and slow down execution speed.
 - When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

```
var x = new String();               // Declares x as a String object
var y = new Number();               // Declares y as a Number object
var z = new Boolean();              // Declares z as a Boolean object
```

JavaScript Strings

- ❑ A JavaScript string stores a series of characters
- ❑ A string can be any text inside “double” or ‘single’ quotes.

```
<script>

    var carName1 = "Volvo XC60";
    var carName2 = 'Volvo XC60';

    document.getElementById("demo").innerHTML =
        carName1 + "<br>" + carName2;

</script>
```

- ❑ You can use quotes inside a string, as long as they don't match the quotes surrounding the string:
 - var answer = "He is called 'Johnny'";

String methods

Method	Description
<u>charAt()</u>	Returns the character at the specified index (position)
<u>charCodeAt()</u>	Returns the Unicode of the character at the specified index
<u>concat()</u>	Joins two or more strings, and returns a new joined strings
<u>endsWith()</u>	Checks whether a string ends with specified string/characters (T or F)
<u>fromCharCode()</u>	Converts Unicode values to characters
<u>includes()</u>	Checks whether a string contains the specified string/characters
<u>indexOf()</u>	Returns the position of the first found occurrence of a specified value in a string
<u>lastIndexOf()</u>	Returns the position of the last found occurrence of a specified value in a string
<u>localeCompare()</u>	Compares two strings in the current locale
<u>match()</u>	Searches a string for a match against a regular expression, and returns the matches
<u>repeat()</u>	Returns a new string with a specified number of copies of an existing string

String methods

Method	Description
<u>replace()</u>	Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced
<u>search()</u>	Searches a string for a specified value, or regular expression, and returns the position of the match
<u>slice()</u>	Extracts a part of a string and returns a new string
<u>split()</u>	Splits a string into an array of substrings
<u>startsWith()</u>	Checks whether a string begins with specified characters
<u>substr()</u>	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
<u>substring()</u>	Extracts the characters from a string, between two specified indices
<u>toLowerCase()</u>	Converts a string to lowercase letters
<u>toString()</u>	Returns the value of a String object
<u>toUpperCase()</u>	Converts a string to uppercase letters
<u>trim()</u>	Removes whitespace from both ends of a string
<u>valueOf()</u>	Returns the primitive value of a String object

String HTML Wrapper Methods

- ❑ The HTML wrapper methods return the string wrapped inside the appropriate HTML tag.
- ❑ These are not standard methods, and may not work as expected in all browsers.

Method	Description
<u>big()</u>	Displays a string using a big font
<u>bold()</u>	Displays a string in bold
<u>fontcolor()</u>	Displays a string using a specified color
<u>fontsize()</u>	Displays a string using a specified size
<u>italics()</u>	Displays a string in italic
<u>link()</u>	Displays a string as a hyperlink
<u>small()</u>	Displays a string using a small font
<u>strike()</u>	Displays a string with a strikethrough
<u>sub()</u>	Displays a string as subscript text
<u>sup()</u>	Displays a string as superscript text

Date Object

- ❑ The Date object is used to work with dates and times.
- ❑ Date objects are created with `new Date()`
- ❑ There are four ways of instantiating a date:

```
var d = new Date();
```

```
var d = new Date(milliseconds);
```

```
var d = new Date(dateString);
```

```
var d = new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

- ❑ A JavaScript date can be written
 - as a string: `Sun Aug 06 2017 15:36:38 GMT+0300` (Arab Standard Time)
 - as a number: `1502022998304`
 - Dates written as numbers, specifies the number of milliseconds since January 1, 1970, 00:00:00.

Date Methods

Get Methods

- ❑ Get methods are used for getting a part of a date.

Method	Description
<u>getDate()</u>	Get the day as a number (1-31)
<u>getDay()</u>	Get the weekday as a number (0-6)
<u>getFullYear()</u>	Get the four digit year (yyyy)
<u>getHours()</u>	Get the hour (0-23)
<u>getMilliseconds()</u>	Get the milliseconds (0-999)
<u>getMinutes()</u>	Get the minutes (0-59)
<u>getMonth()</u>	Get the month (0-11)
<u>getSeconds()</u>	Get the seconds (0-59)
<u>getTime()</u>	Get the time (milliseconds since January 1, 1970)

Date Object functions

Method	Description
<u>setHours()</u>	Sets the hour of a date object
<u>setMilliseconds()</u>	Sets the milliseconds of a date object
<u>setMinutes()</u>	Set the minutes of a date object
<u>setMonth()</u>	Sets the month of a date object
<u>setSeconds()</u>	Sets the seconds of a date object
<u>setTime()</u>	Sets a date to a specified number of milliseconds after/before January 1, 1970
<u>setFullYear()</u>	Sets the year of a date object
<u>toDateString()</u>	Converts the date portion of a Date object into a readable string
<u>toString()</u>	Converts a Date object to a string
<u>valueOf()</u>	Returns the primitive value of a Date object as milliseconds

JavaScript Events

Event Handlers

- ❑ **Event Handler** – a segment of codes (usually a function) to be executed when an event occurs
- ❑ Action that occurs, such as a user clicking a link or button, or user entering data in a form textbox
 - HTML events are "**things**" that happen to HTML elements.
 - When JavaScript is used in HTML pages, JavaScript can "**react**" on these events.
- ❑ An HTML event can be something the browser does, or something a user does. Such as
 - An HTML web page has finished loading
 - An HTML input field was changed
 - An HTML button was clicked
 - A user clicks on a link in a page
 -
- ❑ Often, when events happen, you may want to do something.
 - JavaScript lets you execute code when events are detected.

Event Handlers ...

- ❑ HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

```
<button onclick="myFunction();" >Click me!</button>
```

- ❑ JavaScript **functions can be set as event handlers**
 - when you interact with the element, the function will execute
- ❑ **onclick** is just one of many event HTML attributes we'll use
- ❑ but popping up an alert window is disruptive and annoying
 - A better user experience would be to have the message appear on the page...

Event Handlers

❑ Common HTML Events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page
onReset	The form is reset
onUnLoad	The user closes a document or a frame
onResize	A form is resized by the user

Event Handlers

- ❑ Another way is to **use inline scripts**, i.e. put the script inside the event handler itself:

```
<p id="MyTag"
  onclick="document.getElementById('MyTag').style.fontSize='14pt';
  document.getElementById('MyTag').style.fontWeight='bold';
  ('MyTag').style.color='red'"> This is a paragraph
  thdocument.getElementByIdat has its color changed.</p>
```

- ❑ Note


- The **<script> tag is not necessary** in this case
- Quoted values inside the script must be enclosed in **single quotes** (apostrophes) to alternate and differentiate the sets of quote marks.
- The paragraph “MyTag” (containing the script) refers to itself in the script

onclick Event

Example 1: To access a particular element's value

```
<body>
  <input id="FirstNo" type="text" value="10" />
  <input id="SecondNo" type="text" value="20" />
  <input type="button" value=" = " onclick="Subtract()"/>
  <input id="Output" type="text" />

  <script type="text/javascript">
    function Subtract() {
      document.getElementById("Output").value =
        document.getElementById("FirstNo").value -
        document.getElementById("SecondNo").value;
    }
  </script>
</body>
```



onclick Event

Example 2: To change a particular element's style

```
<p>Hello World !</p>
<input type="button" value="Set Text" onclick="TextSize()">

<script type="text/javascript">
    function TextSize() {
        var ReturnedValue = window.confirm("Larger
        text?");
        if (ReturnedValue == true) {
            document.body.style.fontSize = "32pt";
        } else {
            document.body.style.fontSize = "10pt";
        }
    }
</script>
```

Hello World !

Set Text

The page at www.w3schools.com says:

Larger text?

OK

Cancel

Hello World !

Set Text

onMouseOver Event Handler

```
<p id="demo" onmouseover ="ChangeStyle()"> Change the style </p>

<script type="text/javascript">
function ChangeStyle() {
    document.getElementById ("demo").style.fontSize = "18pt";
    document.getElementById("demo").style.fontWeight = "bold";
    document.getElementById("demo").style.color = "blue";
}

</script>
```

Change the style

Change the style

onload Event handler

```
<html>
  <head><title>Onload Event</title>
    <script>
      function handleOnLoad() {
        window.location="https://www.google.com.sa";
      }
    </script>
  </head>

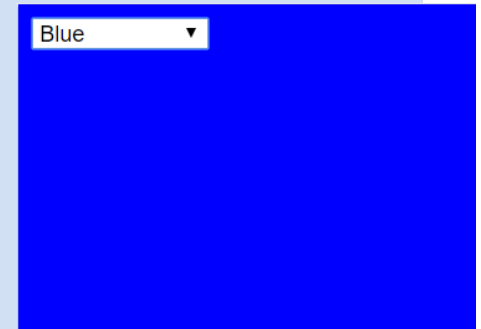
  <body onload="handleOnLoad()">
    <b> Redirecting... </b>
  </body>
</html>
```

Onchange Event Handler

Example 1: change background color

```
<html>
  <head><title>OnChange Event</title>
    <script>
      function handleOnChange(x) {
        document.bgColor=x;
      }
    </script>
  </head>

  <body>
    <select onchange="handleOnChange(this.value)">
      <option value=""> Select a color </option>
      <option value="blue"> Blue </option>
      <option value="green"> Green </option>
    </select>
  </body>
</html>
```



Onchange Event Handler

Example 2: go to a particular website

```
<html>
  <head><title>OnChange Event</title>
    <script>
      function handleChange(x) {
        window.location=x;
      }
    </script>
  </head>

  <body>
    <select onchange="handleChange(this.value)">
      <option value=""> Select a website </option>
      <option value="https://www.google.com.sa"> Google</option>
      <option value="https://www.w3schools.com/"> W3Schools</option>
    </select>
  </body> </html>
```


OnChange Event Handler

Example 3: switch background images

```
<html>
  <head><title>OnChange Event</title>
    <script>
      function handleChange(x) {
        document.body.style.backgroundImage="url('"+x+"')";
      }
    </script>
  </head>

  <body>
    <select onchange="handleChange(this.value)">
      <option value=""> Select an image </option>
      <option value="images/img1.jpg"> Image 1</option>
      <option value="images/img2.jpg"> Image 2 </option>
    </select>
  </body> </html>
```

setTimeout

```
<html>
  <head>
    <script>
      function waitForSeconds() {
        setTimeout( functionName, 2000);
      }
      function functionName() {
        // do some thing
      }
    </script>
  </head>

  <body onload = "waitForSeconds()">
    .....
</body>
</html>
```

setTimeout

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to wait 3 seconds, then alert "Hello".</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  setTimeout(displayAlert, 3000);
}
function displayAlert()
{
  alert("Hello");
}
</script>

</body>
</html>
```

Click the button to wait 3 seconds, then alert "Hello".

Try it

An embedded page on this page says

Hello

OK

User-Defined Objects

User-Defined Objects

- ❑ You can create **complex data structures** by creating your own objects

- JavaScript allows that although it is not a full-featured OO language

```
var person= {firstName:"John", lastName:"Doe", id:556};
```

- The values are written as **name:value** pairs (name and value separated by a **colon**).
 - The **name:values** pairs (in JavaScript objects) are called **properties**

- ❑ You can access object properties in two ways:

- `objectName.propertyName` `person.lastName;`
 - or
 - `objectName["propertyName"]` `person["lastName"];`

User-Defined Objects..

- ❑ Adding functions to objects:

```
var person = {  
    firstName: "John",  
    lastName : "Doe",  
    id      : 556,  
  
    fullName : function() {  
        return this.firstName + " " + this.lastName;  
    }  
};
```

- ❑ You **access an object method** with the following syntax:
 - `objectName.methodName()`

Example

```
document.getElementById("demo").innerHTML = person.fullName();
```

Create object using **Literal** Constructor

```
<script type="text/javascript">

    var blogpost ={
        title:"JavaScript",
        text: "It is a powerful !",

        displayDetails: function(){
            document.write(this.title + "<br/>");
            document.write(this.text);
        }
    };

    blogpost.displayDetails();

</script>
```

Object Creation using **new** Keyword

```
<html>

<head> <script type="text/javascript">
    var blogpost =new Object();
    blogpost.title="JavaScript";
    blogpost.text="It is a powerful !";
    blogpost.category="UI Technology";

    document.write(blogpost.title + "<br/>");
    document.write(blogpost.text + "<br/>");
    document.write(blogpost.category);

</script> </head>

<body> </body> </html>
```


Object Creation using **Function**..

```
<html>
<head> <script type="text/javascript">
    function blogpost(title, text){
        this.title = title;
        this.text= text;
    }

    var bp1= new blogpost("JavaScript", "It is a powerful !");
    document.write(bp1.title + "<br/>");
    document.write(bp1.text + "<br/>");

    var bp2= new blogpost("jQuery", "Makes JS much easier!");
    document.write(bp2.title + "<br/>");
    document.write(bp2.text + "<br/>");

</script> </head>
<body> </body> </html>
```

JavaScript Forms

You can start by HTML forms then go with JS
enhancement

The Form Object

- ❑ **Form** is an **area in your web-page** which can contain **form elements**.
 - This area is specified by `<FORM></FORM>` tags.
- ❑ Form-elements are elements that **allow the web-page to get data from user** by providing graphical interfaces to the users to enter their data
- ❑ Any standard HTML element (except another `<form>`) can be contained within `<form>`
 - **Attributes:** **NAME** = “name” and **ACTION** =“url”
 - **NAME:** Name the form (For tasks related to user input data processing)
 - **ACTION:** The URL of the program that will process the data when the form is submitted

Processing Data

- ❑ After the **web page receives the data (input) from the client** of the web page by some means (such as a form elements), **it needs to be processed**.
- ❑ There are two possible locations for processing data
 - The data can be sent thru network to a web-server machine where all such processing requests are processed. Then the result will be sent to the client machine which initiated the request by opening a page and filling some sort of forms.
 - The data can be processed at client machine by some tiny programs embedded in the web page. These tiny programs are sent to client side along the web page when the user loads the web page.

Validating Form Data

- ❑ JavaScript can be used to **validate input data in HTML forms** before sending off the content to a server.
- ❑ Form data are typically checked by a JavaScript to see, e.g. :
 - has the user left **required fields** empty?
 - has the user entered a **valid e-mail** address?
 - has the user entered a **valid date**?
 - has the user entered text in **a numeric field**?

Form Validation

```
<body>
```

```
<h2> User Registration: </h2>
```

```
<form name="myform" action="login.php" onsubmit= "return validate()">
```

```
  Email: <input type="text" name="email"/>
```

```
  Password: <input type="password" name="password"/>
```

```
  <input type="submit" value="Register"/>
```

```
</form>
```

```
<script type="text/javascript">
```

```
  function validate() {
```

```
    var email=document.myform.email.value;
```

```
    var psw=document.myform.password.value;
```

```
    if (email==""){
```

```
      alert("Email is mandatory!");
```

```
      return false;
```

```
    } else if (psw==""){
```

```
      alert("Password is mandatory!");
```

```
      return false;
```

```
    }else {
```

```
      alert("Successful Validation !");
```

```
      return true; }}
```

```
</script> </body>
```

Validate dropdown list

```
<head> <script type="text/javascript">
function validate(){
    var color = document.myform.color.value;
    if (color==""){
        alert("Please select a color");
        document.myform.color.focus();
    }
}
</script> </head>
```

What is missing?

```
<body>
<h2> Select a Color : </h2>
<form name="myform" action="dropdown.php" onsubmit= "return validate()">
    <select name="color">
        <option value=""> Select a color </option>
        <option value="blue"> Blue </option>
        <option value="green"> Green </option> </select>
        <input type="submit" value="Submit"/>
    </form> </body>
```

Validate checkbox

```
<head> <script type="text/javascript">
function validate() {
    var valid=false;
    if(document.getElementById("javascript").checked) {
        valid=true;
    }else if(document.getElementById("jquery").checked) {
        valid=true; }
    if (valid){
        return true;
    }else {
        alert("please select at least one technology");
        return false;} }
</script> </head>

<body>
<h2> Select a Technology: </h2>
<form name="myform" action="checkboxes.php" onsubmit= "return validate()">
    <input type="checkbox" id="javascript" value="JavaScript"> JavaScript
    <input type="checkbox" id="jquery" value="jQuery"> jQuery
    <input type="submit" value="Submit"/>
</form> </body>
```


Validating Radio Button

```
<head> <script type="text/javascript">
    function validateForm() {
        if( document.forms["survey1"]["q1"].checked)
        {    return true;
        } else {
            alert('Please answer all questions');
            return false; } }
</script> </head>
```

```
<body>
<form name="survey1" action="add5up.php" method="post"
    onsubmit="return validateForm()">
<div id="question">Q1) My programme meets my expectations</div><br />
Always<INPUT  TYPE = 'Radio' Name ='q1'  value= 'a'>
Usually<INPUT  TYPE = 'Radio' Name ='q1' value= 'b'>
Rarely<INPUT  TYPE = 'Radio' Name ='q1' value= 'c'>
Never<INPUT  TYPE = 'Radio' Name ='q1' value= 'd'>
<input type="submit" value="addData" />
</form>
</body>
```

Regular Expression

- ❑ A regular expression is an object that describes **a pattern of characters**.
 - almost the same as in Perl or Java
- ❑ **Regular expressions** are patterns used to **match character combinations** in strings. A regular expression can be constructed **statically** or **dynamically**
 - Within **slashes**, such as `re = /ab+c/`
 - With a **constructor**, such as `re = new RegExp("ab+c")`
 - Used when the pattern to match is taken as user input

- ❑ Examples

- `var pattern = /[0-9]/` >> **matches a digit**
- `var pattern = /[A-Z a-z]/` >> **Matches a letter**

Abc#1234

username@gmail.com

- ❑ It can specify more complex regular expressions

- E.g. match phone numbers, email addresses, url, etc

Validate the given input.. ?

Validation Using Regular Expression

```
var pattern = /pattern/attributes;  
  
var pattern = new RegExp (pattern, attributes);  
  
pattern.test(" ");
```

RegExp Object...

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

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x y)	Find any of the alternatives specified

Validation Using Regular Expression

❑ Quantifiers

Quantifier	Description
<u>n^+</u>	Matches any string that contains at least one n
<u>n^*</u>	Matches any string that contains zero or more occurrences of n
<u>$n^?$</u>	Matches any string that contains zero or one occurrences of n
<u>$n\{X\}$</u>	Matches any string that contains a sequence of X n 's
<u>$n\{X,Y\}$</u>	Matches any string that contains a sequence of X to Y n 's
<u>$n\{X, \}$</u>	Matches any string that contains a sequence of at least X n 's
<u>$n\\$</u>	Matches any string with n at the end of it
<u>n</u>	Matches any string with n at the beginning of it

Validation Using Regular Expression

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

Metacharacter	Description
.	Find a single character, except newline or line terminator
<u>\s</u>	Find a whitespace character
<u>\S</u>	Find a non-whitespace character
<u>\d</u>	Find a digit
<u>\D</u>	Find a non-digit character
<u>\b</u>	Find a match at the beginning/end of a word
<u>\B</u>	Find a match not at the beginning/end of a word
<u>\0</u>	Find a NUL character
<u>\n</u>	Find a new line character
.....	

JavaScript String match() Method

- ❑ The `match()` method searches a string for a match against a regular expression, and returns the matches, as an **Array object**.
- ❑ This method returns *null* if no match is found.

- Example:

```
var str = "The rain in SPAIN stays mainly in the plain";  
var res = str.match(/ain/g);
```

- Output: ain, **ain**, **ain**

Perform a global match (find all matches rather than stopping after the first match)

JavaScript String test() Method

- ❑ The **test()** method tests for a match in a string.
- ❑ This method returns **true if it finds a match**, otherwise it returns false.
- ❑ Example:

```
var str = "The best things in life are free";  
var patt = new RegExp("e");  
var res = patt.test(str);
```

- **Output:** Variable res will be *true*

Tests for a match in a string.
Returns true or false

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_regexp_test2
https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_regexp_test

JavaScript exec() Method

- ❑ The exec() method tests for a match in a string.
- ❑ This method **returns the matched text if it finds a match**, otherwise it returns null.
- ❑ Example:

```
var str = "The best things in life are free";  
var patt = new RegExp("e");  
var res = patt.exec(str);
```

- **Output: e**

Tests for a match in a string.
Returns the first match

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_regexp_exec2
https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_regexp_exec

Examples

- ❑ **\b** : used to find a match at the **beginning or end of a word**. If no match is found, it returns null.

```
var str = "Visit W3Schools";  
var patt1 = /\bW3/;  
var result = str.match(patt1); => W3
```

- ❑ ***/\bregex/g***

```
var str = "Visit W3Schools W3test";  
var patt1 = /\bW3/g;  
var result = str.match(patt1); => W3, W3
```

Examples..

- ❑ **\w**: used to find a **word character**. A word character is a character from a-z, A-Z, 0-9, including the _ (underscore) character.



```
var str = "Give 100%!";  
var patt1 = /\w/g;  
var result = str.match(patt1); => G,i,v,e,1,0,0
```

- ❑ **^**: used to find a match at **the beginning or end of a string**. If no match is found, it returns null.

```
var str = "Is this his";  
var patt1 = /^Is/g;  
var result = str.match(patt1); => Is
```

Example 1

```
<head> <script type="text/javascript">
    function validate() {
        var exp=/^[A-Za-z]+$ /;
        var username =document.myform.uname.value;
        var result= exp.test(username) ;
        if (result){
            alert("Validation successful");
        }else{
            alert("Validation failed");
            return false;
        }
    }
</script> </head>

<body>
    <h2> User Registration: </h2>
    <form name="myform" action="reg.php" onsubmit= "return
validate() ">
        User Name: <input type="text" name="uname" />
        <input type="submit" value="Submit" />
    </form> </body>
```

Example 2

```
var exp=/^[A-Za-z0-9\s]{3,10}$/;
```

Specify number of characters

```
<head> <script type="text/javascript">
function validate() {
    var exp=/^[A-Za-z0-9\s]+$/;
    var address =document.myform.address.value;
    var result= exp.test(address);
    if (result){
        alert("Validation successful");
    }else{
        document.getElementById("err").innerHTML="Please
        enter alpha numeric values only";
        document.myform.address.focus();
        return false;
    }
}
</script> </head> <body>
<h2> User Registration: </h2>
<form name="myform" action="registration.php" onsubmit=
"return validate()">
    Address: <textarea name="address"></textarea>
    <input type="submit" value="Submit"/>
    <span id="err"></span>
</form> </body>
```

Validate email address using a regex

- ❑ JavaScript to validate email address using a regex

```
/^[A-Za-z0-9_\-\.]+\@([A-Za-z0-9_\-\.]+\.[A-Za-z]{2,4})$/
```

Exception handling

Exception handling

- ❑ Exception handling in JavaScript is *almost the same as in Java*
- ❑ **throw** *expression* creates and throws an exception
 - The *expression* is the value of the exception, and can be of *any* type (often, it's a literal String)

```
try {  
    statements to try  
} catch (e) {      // Notice: no type declaration for e  
    exception handling statements  
} finally {       // optional, as usual  
    code that is always executed  
}
```

- With this form, there is *only one* **catch** clause

Exception handling

```
try {  
    statements to try  
} catch (e if test1) {  
    exception handling for the case that test1 is true  
} catch (e if test2) {  
    exception handling for when test1 is false and test2  
is true  
} catch (e) {  
    exception handling for when both test1 and test2 are  
false  
} finally {           // optional, as usual  
    code that is always executed  
}
```

- ❑ Typically, the test would be something like
 e == "InvalidNameException"

JavaScript Debugging

- ❑ Programming code might contain **syntax errors**, or **logical errors**.
- ❑ Many of these errors are difficult to diagnose !!
- ❑ **Code Debugging**- Searching for (and fixing) errors in programming code is called
- ❑ All modern browsers have a **built-in JavaScript debugger**.
 - Built-in debuggers can be turned on and off, forcing errors to be reported to the user.
- ❑ With a debugger, you can also set **breakpoints** (places where code execution can be stopped), and examine variables while the code is executing.

