JAVA SCRIPT

JavaScript

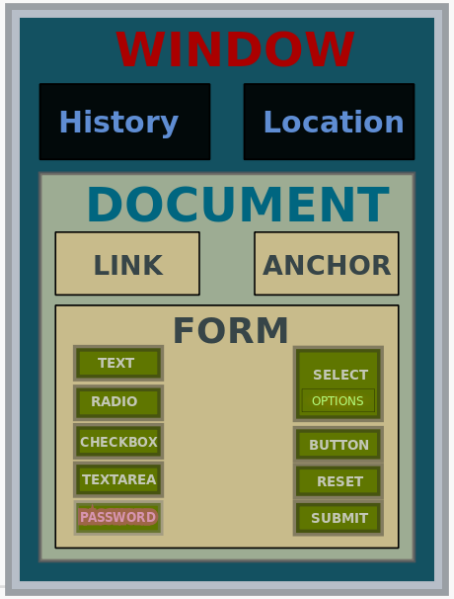
- JavaScript was originally developed by Brendan Eich, while working for Netscape Communications Corporation
- Initially JavaScript was developed under the name *Mocha*, the language was officially called *LiveScript* when it first shipped in beta releases of Netscape Navigator 2.0 in September 1995
- Bsut it was renamed JavaScript,\when it was deployed in the Netscape browser version 2.0.

- JavaScript is a scripting language developed by Netscape.
- JavaScript works in all major browsers that are version 3.0 or higher.
  JavaScript is supported by all major browsers like Netscape and Internet Explorer.
- The JavaScript can be
- Immediate script:
  - *executed immediately*
  - *Scripts in the body section will be executed while the page loads.*
- Deferred script:
  - *executed at a later event.*
  - *Scripts in the head section will be executed when called.*

DOM (Document Object Model)

- The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.
- The document can be further processed and the results of that processing can be incorporated back into the presented page.
- DOM is an Object Hierarchy
- DOM address: A DOM element equivalent to an HTML element , that has a dot decimal address

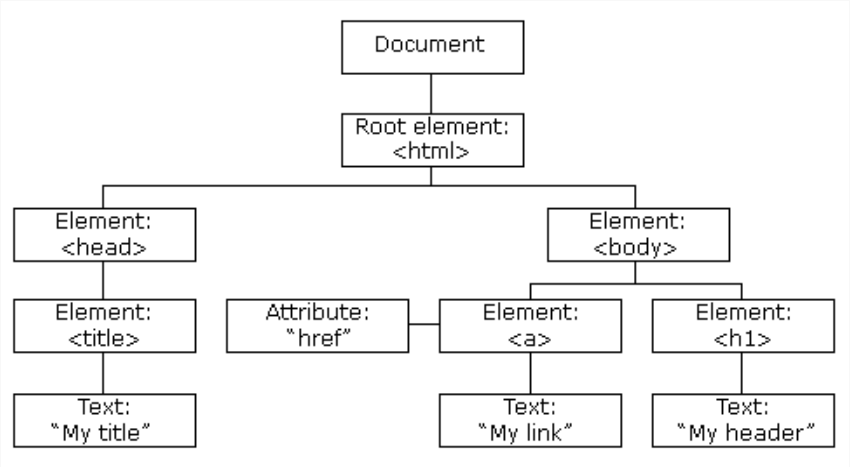
window.document.form[0].rollNo.value



Source: http://en.wikipedia.org/wiki/Document_Object_Model

The HTML DOM

- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The **HTML DOM** model is constructed as a tree of **Objects:**



- With a programmable object model, JavaScript gets all the power it needs to create dynamic HTML:
- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can react to all the events in the page

Finding HTML Elements

- To manipulate HTML elements, we have to find the elements first.
- There are a couple of ways to do this:
  - *Finding HTML elements by id*
  - *Finding HTML elements by tag name*
  - *Finding HTML elements by class name*

Finding HTML Elements by Id

- The easiest way to find HTML elements in the DOM, is by using the element id.
- **Example**

```
var x=document.getElementById("intro");
```

- This example finds the element with id="intro":
- If the element is found, the method will return the element as an object (in x).
- If the element is not found, x will contain null.

Example

Finding HTML Elements by Tag Name

- **Example**

```
var x=document.getElementById("main");

var y=x.getElementsByTagName("p");
```

- This example finds the element with id="main", and then finds all <p> elements inside "main":

Example

Example

Finding HTML Elements by Class Name

- Example

```
<div id="test">
<p class="one"></p>
<p class="two"></p>
< p class="three"></p >
<p class="four"></p>
</div>
```

```
doc = document.getElementById('test');
notes = doc.getElementsByClassName('four');
```

- Finding elements by class name does not work in Internet Explorer 5,6,7, and 8.

Example

Changing the HTML Output Stream

- JavaScript can create dynamic HTML content:
- In JavaScript, document.write() can be used to write directly to the HTML output stream:

Example

```
<html>
    <body>

    <script>
                document.write(Date());
    </script>

    </body>
    </html>
```

Changing HTML Content

- The easiest way to modify the content of an HTML element is by using the **innerHTML** property.
- Syntax:

document.getElementById(*id*).innerHTML=*new HTML*

- This example changes the content of a <p> element:
- **Example**
-

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
        document.getElementById("p1").innerHTML="New text!";
</script>

</body>
</html>
```

Example

Changing an HTML Attribute

- To change the attribute of an HTML element, use this syntax:

  document.getElementById(*id*).*attribute=new value*

- This example changes the src attribute of an <img> element:
- **Example**
- <!DOCTYPE html>
  <html>
  <body>

  <img id="image" src="IST.jpg">

  <script>
  document.getElementById("image").src="img.jpg";
  </script>

  </body>
  </html>

Example

Changing HTML Style

- To change the style of an HTML element, use this syntax:

document.getElementById(*id*).style.*property=new style*

- The following example changes the style of a <p> element:

Example

```
<html>
<body>
<p id="p1">Hello World!</p>
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color="blue";
document.getElementById("p2").style.fontFamily="Arial";
document.getElementById("p2").style.fontSize="larger";
</script>
<p>The paragraph above was changed by a script.</p>
</body>
</html>
```

Example

Reacting to Events

- HTML DOM allows JavaScript to react to HTML events.
- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=*JavaScript*

- Examples of HTML events:
- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

onclick

- **Example**
- <!DOCTYPE html>
  <html>
  <body>
  <h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>
  </body>
  </html>

Example

Assign Events Using the HTML DOM

- The HTML DOM allows you to assign events to HTML elements using JavaScript:

Example

Examples

- onload Example
- onchange Example
- onmouseover and onmouseout Example
- onmousedown and onmouseup Example
- onfocus Example
- onblur Example

What can a JavaScript Do?

- **JavaScript gives HTML designers a programming tool**
  - *HTML authors are normally not programmers, but JavaScript is a very light programming language with a very simple syntax! Almost anyone can start putting small "snippets" of code into their HTML documents.*

- **JavaScript can put dynamic text into an HTML page**
  - *A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into the display of an HTML page, just like the static HTML text: <h1>Bill Gates</h1> does.*

Contd..

- **JavaScript can react to events**
  - *A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element.*
- **JavaScript can read and write HTML elements**
  - *A JavaScript can read an HTML element and change the content of an HTML element.*
- **JavaScript can be used to validate data**
  - *JavaScripts can be used to validate data in a form before it is submitted to a server.*
  - *This function is particularly well suited to save the server from extra processing.*

Example

```
<html>
<body>
<center>
<h4>
This page was last modified on
<script type="text/javascript ">
document.write(document.lastModified)
</script>
</h4>
</center>
</body>
</html>
```

Example

How to Put a JavaScript Into an HTML Document

- To insert a script in an HTML document, use the <script> tag.
- Use the type attribute to define the scripting language.

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

- Then comes the JavaScript: In JavaScript the command for writing some text on a page is **document.write**:

```
document.write("Hello World!")
```

- The script ends:

```
</script>
```

Where to Put the JavaScript

- **Scripts in the head section:** Scripts to be executed when they are called, or when an event is triggered, go in the head section.

```
<html>
<head>
<script type="text/javascript"> some statements </script>
</head>
```

- **Scripts in the body section:** Scripts to be executed when the page loads go in the body section. When you place a script in the body section it generates the content of the page.

```
<html>
<head>
</head>
<body>
<script type="text/javascript"> some statements </script>
</body>
```

Contd..

- **Scripts in both the body and the head section:** We can place an unlimited number of scripts in our document, so we can have scripts in both the body and the head section.

```
<html>
<head>
<script type="text/javascript"> some statements </script>
</head>
 <body>
<script type="text/javascript"> some statements </script>
</body>
```

External JavaScript

- Sometimes we need to run the same script on several pages, without writing the script on each and every page.
- To simplify this we can write a script in an external file, and save it with a **.js** file extension
- The external script cannot contain the <script> tag
- Now we can call this script, using the "src" attribute, from any of our pages
- Place the script exactly where we normally would write the script.

Example

JavaScript Guidelines

- **JavaScript is Case Sensitive**
  - *A function named "myfunction" is not the same as "myFunction".*
- **Symbols**
  - *Open symbols, like ({ [" ', must have a matching closing symbol, like ' "] }).*
- **White Space**
  - *JavaScript ignores extra spaces. These two lines mean exactly the same:*

```
name=“IT"
name   =   “IT“
```

Contd..

- **Insert Special Characters**
  - *We can insert special characters (like " ' ; &) with the backslash:*

document.write (“You \& I will \“go to University\”.”)

The example above will produce this output:

You & I will “go to University”.

- **Comments**
  - *We can add a comment to our JavaScript code starting the comment with two slashes "//":*

sum=a + b //calculating the sum

- We can also add a comment to the JavaScript code, starting the comment with "/*" and ending it with "*/"

sum=a + b /*calculating the sum*/

- Using "/*" and "*/" is the only way to create a multi-line comment:

JavaScript Variables

- **Rules for Variable names:**
  - *Variable names are case sensitive*
  - *They must begin with a letter or the underscore character*
- **Declare a Variable**
  - *We can create a variable with the **var** statement:*

    var strname = *some value*
  - *We can also create a variable **without** the **var** statement:*

    strname = *some value*
- **Assign a Value to a Variable**
  - *We assign a value to a variable like this:*

    var strname = “IT”
  - *Or like this:*

    strname = “IT”

Lifetime of Variables

- When we declare a variable within a function, the variable can only be accessed within that function.
- When we exit the function, the variable is destroyed. These variables are called local variables.
- we can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.
- If we declare a variable outside a function, all the functions on our page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

JavaScript Operators

- **Arithmetic Operators**

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operators

Operator	Example	Is Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Comparison Operators

- If **x = 5;**

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
===	equal value and equal type	x === "5"	false
		x === 5	true
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== "5"	true
		x !== 5	false
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	*true*

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 x != y returns true

JavaScript Bitwise Operators

Operator	Description	Example	Same as	Result	Decimal
&	AND	x = 5 & 1	0101 & 0001	0001	1
|	OR	x = 5 | 1	0101 | 0001	0101	5
~	NOT	x = ~ 5	~0101	1010	10
^	XOR	x = 5 ^ 1	0101 ^ 0001	0100	4
<<	Left shift	x = 5 << 1	0101 << 1	1010	10
>>	Right shift	x = 5 >> 1	0101 >> 1	0010	2

Above examples uses 4 bits unsigned values. But bitwise operator treat their operands as a sequence of 32 bit signed numbers.
Because of this , ~ 5 will not return 10. It will return -6.

•~00000000000000000000000000000101 will return 11111111111111111111111111111010

String Operator

- A string is most often a text, for example "Hello World!". To stick two or more string variables together, use the + operator.

  txt1="What a very"
  txt2="nice day!"
  txt3=txt1+txt2

- The variable txt3 now contains "What a very nice day!".
- To add a space between two string variables, insert a space into the expression, OR in one of the strings.

  txt1="What a very"
  txt2="nice day!"
  txt3=txt1+" "+txt2 or txt1="What a very "
  txt2="nice day!"
  txt3=txt1+txt2

- The variable txt3 now contains "What a very nice day!".

Function

- **Define a Function**

– *To create a function we define its name, any values ("arguments"), and required statements:*

```
function myfunction(argument1,argument2,etc)
  {
      some statements
  }
```

- A function with no arguments must include the parentheses:

```
function myfunction()
  {
      some statements
  }
```

Contd..

- Place the function code in the head section of the document
- Some functions return a value to the calling expression

```
function result(a, b)
  {
  c=a+b
  return c
  }
```

To Call a Function

- A function is not executed before it is called.
- *Function call syntax:*

myfunction(argument1,argument2,etc)

- or without arguments:

myfunction()

- **The return Statement**
- *Functions that will return a result must use the "return" statement.*

```
function total(a,b)
 {
        result=a+b
        return result
 }
```

- For this function we must send two arguments with it:

sum=total(2,3)

- The returned value from the function (5) will be stored in the variable called sum.

Conditional Statements

- In JavaScript we have three conditional statements:
- ***if statement*** *- use this statement if you want to execute a set of code when a condition is true*
- ***if...else statement*** *- use this statement if you want to select one of two sets of lines to execute*
- ***switch statement*** *- use this statement if you want to select one of many sets of lines to execute*

If and If...else Statement

```
if (condition)
{
        code to be executed if condition is true
 }
 else
        {
                code to be executed if condition is false
        }
```

Example

Switch Statement

- **Syntax**

```
switch (expression)
{
        case label1: code to be executed if expression = label1
                        break
        case label2: code to be executed if expression = label2
                        break
        default: code to be executed
}
```

Example

Conditional Operator

- JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.
- **Syntax**

Variable name=(condition)?value1:value2

- **Example**

greeting=(visitor==“IST")?“My Dear Student":“ Hello outsider”

- If the variable visitor is equal to IST, then put the string “My Dear student " in the variable named greeting.
- If the variable visitor is not equal to IST, then put the string “Hello outsider " into the variable named greeting.

Example

JavaScript Looping

- Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to do this.
- In JavaScript we have the following looping statements:
  - ***while*** *- loops through a block of code while a condition is true*
  - ***do...while*** *- loops through a block of code once, and then repeats the loop while a condition is true*
  - ***for*** *- run statements a specified number of times*

while

- The while statement will execute a block of code while a condition is true..

```
while (condition)
      {
      code to be executed
   }
```

do...while

- The do...while statement will execute a block of code once, and then it will repeat the loop while a condition is true

```
do
    {
    code to be executed
    }
    while (condition);
```

for

- The for statement will execute a block of code a specified number of times

```
for (initialization; condition; increment)
    {
      code to be executed
    }
```

Example

JavaScript Objects

- JavaScript has several **built-in** objects, like String, Date, Array, and more.
- An object is just a special kind of data, with **properties** and **methods.**
- **Accessing Object Properties**

– *Properties are the values associated with an object.*

– *The syntax for accessing the property of an object is:*

objectName.propertyName

– *This example uses the length property of the String object to find the length of a string:*

```
var message="Hello World!";
var x=message.length;
```

– *The value of x, after execution of the code above will be: 12*

Accessing Objects Methods

- Methods are the actions that can be performed on objects.
- We can call a method with the following syntax:

objectName.methodName()

- This example uses the toUpperCase() method of the String object, to convert a text to uppercase:

```
var message="Hello world!";
var x=message.toUpperCase();
```

- The value of x, after execution of the code above will be: HELLO WORLD!

Creating JavaScript Objects

- With JavaScript we can define and create our own objects.
- There are 2 different ways to create a new object:
  - *Define and create a direct instance of an object.*
  - *Use a function to define an object, then create new object instances.*

Creating a Direct Instance

- This example creates a new instance of an object, and adds four properties to it:
- **Example**

```
person=new Object();
person.firstname=“Ram";
person.lastname=“Das";
person.age=50;
person.eyecolor="blue";
```

- Alternative syntax (using object literals):

```
person={firstname:“Ram",lastname:"Das",age:50,eyecolor:"blue"};
```

Example

Using an Object Constructor

- This example uses a function to construct the object:
- **Example**

```
function person(firstname,lastname,age,eyecolor)
{
        this.firstname=firstname;
        this.lastname=lastname;
        this.age=age;
        this.eyecolor=eyecolor;
}
```

Example

JavaScript String Object

- A string can be any text inside quotes.
- We can use single or double quotes.
- **String Length**

– *The length of a string (a string object) is found in the built in property* ***length****:*

```
var  txt="Hello World!";
document.write(txt.length);
```

Contd..

- **Finding a String in a String**
  - *The **indexOf()** method returns the position (as a number) of the first found occurrence of a specified text inside a string:*
  - ***Example***

```
var str="Hello world, welcome to the universe.";
var n=str.indexOf("welcome");
```

- **Matching Content**
  - *The **match()** method can be used to search for a matching content in a string:*
  - ***Example***

```
var str="Hello world!";
document.write(str.match("world") + "<br>");
```

Example

Contd..

- **Replacing Content**
  - *The **replace()** method replaces a specified value with another value in a string.*
  - ***Example***

```
str="Please visit Microsoft!"
var n=str.replace("Microsoft",“Ubuntu");
```

- **Upper Case and Lower Case**
  - *A string is converted to upper/lower case with the methods **toUpperCase()** / **toLowerCase():***
  - ***Example***

```
var txt="Hello World!";       // String
var txt1=txt.toUpperCase();   // txt1 is txt converted to upper
var txt2=txt.toLowerCase();   // txt2 is txt converted to lower
```

Example

Contd..

- **Convert a String to an Array**
  - *A string is converted to an array with the built in method string.**split():***
  - ***Example***

```
txt="a,b,c,d,e"   // String
txt.split(",");   // Split on commas
txt.split(" ");   // Split on spaces
txt.split("|");   // Split on pipe
```

Example

String Object Methods

Method	Description
charAt()	Returns the character at the specified index
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings
fromCharCode()	Converts Unicode values to characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
search()	Searches for a match between a regular expression and a string, and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
toLowerCase()	Converts a string to lowercase letters
toUpperCase()	Converts a string to uppercase letters
valueOf()	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.

Method	Description
anchor()	Creates an anchor
big()	Displays a string using a big font
blink()	Displays a blinking string
bold()	Displays a string in bold
fixed()	Displays a string using a fixed-pitch font
fontcolor()	Displays a string using a specified color
fontsize()	Displays a string using a specified size
italics()	Displays a string in italic
link()	Displays a string as a hyperlink
small()	Displays a string using a small font
strike()	Displays a string with a strikethrough
sub()	Displays a string as subscript text
sup()	Displays a string as superscript text

JavaScript Array Object

- An Array object is used to store a set of values in a single variable name. Each value is an element of the array and has an associated index number.
- We can refer to a particular element in the array by using the name of the array and the index number.
- The index number starts at zero.
- We create an instance of the Array object with the "new" keyword.
- **Syntax:**

Var array_name=new Array(array size)

Contd..

```
var family_names=new Array(5)
```

- The expected number of elements goes inside the parentheses, in this case 5.
- We can assign data to each of the elements in the array like this:

```
family_names[0]="A"
  family_names[1]="B"
  family_names[2]="C"
  family_names[3]="D"
  family_names[4]="E"
```

- And the data can be retrieved from any element by using the index of the particular array element we want. Like this:

```
mother=family_names[0]
   father=family_names[1]
```

JavaScript Date Object

- The Date object is used to work with dates and times.
- **Examples**
- **Date :**Returns today's date including date, month, and year. Note that the getMonth method returns 0 in January, 1 in February etc. So add 1 to the getMonth method to display the correct date.
- **Time :**Returns the current local time including hour, minutes, and seconds. To return the GMT time use getUTCHours, getUTCMinutes etc.
- **Set date :** We can also set the date or time into the date object, with the setDate, setHour etc. Note that in this example, only the FullYear is set.
- **UTC time** :The getUTC Date method returns the Universal Coordinated Time which is the time set by the World Time Standard.
- **Display weekday :**Gives week days number starting from 0 for Sunday. Sunday=0, Monday=1 etc.
-

Example

Contd..

- The Date object is used to work with dates and times.
- We can create an instance of the Date object with the "new" keyword.
- To store the current date in a variable called "my_date":

```
var my_date=new Date()
```

- After creating an instance of the Date object, we can access all the methods of the object from the "my_date" variable.
- for example, if we want to return the date (from 1-31) of a Date object, we should write the following:

```
my_date.getDate()
```

- We can also write a date inside the parentheses of the Date() object, like this:

```
new Date("Month dd, yyyy hh:mm:ss")
new Date("Month dd, yyyy")
new Date(yy,mm,dd,hh,mm,ss)
new Date(yy,mm,dd)
new Date(milliseconds)
```

Contd..

```
var my_date=new Date("October 12, 1988 13:14:00")
var my_date=new Date("October 12, 1988")
var my_date=new Date(88,09,12,13,14,00)
var my_date=new Date(88,09,12)
var my_date=new Date(500)
```

Contd..

Methods	Explanation
Date()	Returns a Date object
getDate()	Returns the date of a Date object (from 1-31)
getDay()	Returns the day of a Date object (from 0-6. 0=Sunday, 1=Monday, etc.)
getMonth()	Returns the month of a Date object (from 0-11. 0=January, 1=February, etc.)
getFullYear()	Returns the year of a Date object (four digits)
getYear()	Returns the year of a Date object (from 0-99). Use getFullYear instead !!
getHours()	Returns the hour of a Date object (from 0-23)
getMinutes()	Returns the minute of a Date object (from 0-59)
getSeconds()	Returns the second of a Date object (from 0-59)
getMilliseconds()	Returns the millisecond of a Date object (from 0-999)
getTime()	Returns the number of milliseconds since midnight 1/1-1970
getTimezoneOffset()	Returns the time difference between the user's computer and GMT

JavaScript Math Object

- The built-in Math object includes mathematical constants and functions.
- **Examples**
- **Round :**Round a specified number to the nearest whole number

```
r_number=Math.round(8.6)
```

- **Random number :**Random method returns a random number between 0 and 1

```
r_number=Math.random()
```

- **Random number from a to b :**Random number from a to b using the random method.

```
Math.floor(Math.random() * (b - a + 1)) + a;
```

- **Max number :**Test which of two numbers, has the higher value.

```
Math.max(0, 150, 30, 20, -8);      // returns 150
```

- **Min number :**Test which of two numbers, has the lower value.

```
Math.min(0, 150, 30, 20, -8);      // returns -8
```

Example

Contd..

- **Math.ceil():** Math.ceil() rounds a number **up** to the nearest integer:

```
Math.ceil(4.4);          // returns 5
```

- **Math.floor():** Math.floor() rounds a number **down** to the nearest integer:

```
Math.floor(4.7);         // returns 4
```

JavaScript Number Object

- JavaScript has only one type of number.
- JavaScript numbers can be written with, or without decimals:
- **Example**

`

```
var pi=3.14;    // Written with decimals
var x=34;       // Written without decimals
```

- Extra large or extra small numbers can be written with scientific (exponent) notation:
- **Example**

```
var y=123e5;    // 12300000
var z=123e-5;   // 0.00123
```

- **JavaScript Numbers are Always 64-bit Floating Point**

– *JavaScript is not a typed language. Unlike many other programming languages, it does not define different types of numbers, like integers, short, long, floating-point etc.*

– *JavaScript numbers are always stored as double precision floating point numbers*

– *This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:*

Value (Mantissa)	Exponent	Sign
52 bits (0 - 51)	11 bits (52 - 62)	1 bit (63)

Contd..

- **Precision:**Integers (numbers without a period or exponent notation) are considered accurate up to 15 digits.
- **Example**

```
var x = 999999999999999;   // x will be 999999999999999
var y = 9999999999999999;  // y will be 10000000000000000
```

- Floating point arithmetic is not always 100% accurate:
- **Example**

```
var x = 0.2 + 0.1;         // x will be 0.30000000000000004
```

- To solve the problem above we can do it by following way
- **Example**

```
var x = (0.2 * 10 + 0.1 * 10) / 10;       // x will be 0.3
```

Example

Contd..

- **Hexadecimal**
  - *JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.*
- **Example**

```
var x = 0xFF;          // x will be 255
```

- Never write a number with a leading zero (like 07). Some JavaScript versions interpret numbers as octal if they are written with a leading zero , like **var y=0377;**

Contd..

- By default, Javascript displays numbers as base 10 decimals.
- But we can use the toString() method to output numbers as base 16 (hex), base 8 (octal), or base 2 (binary).
- **Example**

```
var myNumber = 128;
myNumber.toString(16);    // returns 80
myNumber.toString(8);     // returns 200
myNumber.toString(2);     // returns 10000000
```

Contd..

- **NaN - Not a Number :**NaN is a JavaScript reserved word indicating that a value is not a number. Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number):
- **Example**

```
var x = 100 / "Apple";  // x will be NaN (Not a Number)
```

- However, if the string contains a numeric value , the result will be a number:
- **Example**

```
var x = 100 / "10";     // x will be 10
```

- We can use the global JavaScript function isNaN() to find out if a value is a number.
- **Example**

```
var x = 100 / "Apple";
isNaN(x);               // returns true because x is Not a Number
```

- NaN is a number: typeOf NaN returns number.
- **Example**

```
typeof NaN;             // returns "number"
```

Numbers Can be Objects

- Normally JavaScript numbers are primitive values created from literals: **var x = 123**
- But numbers can also be defined as objects with the keyword new:

var y = new Number(123)

- **Example**

```
var x = 123;
var y = new Number(123);

typeof  x;          // returns number
typeof  y;          // returns object
```

- **Example**

```
var x = 123;
 var y = new Number(123);
 (x = = =y) // is false because x is a number and y is an object.
```

Number Properties and Methods

- **Properties:**
- *MAX _VALUE: Returns the largest number possible in JavaScript*
- Eg: document.getElementById("demo").innerHTML =Number.MAX_VALUE;
- *MIN_VALUE : Return the smallest number possible in JavaScript:*
- Eg: document.getElementById("demo1").innerHTML=Number.MIN_VALUE;

Contd..

- **Methods:**
  - *toExponential() : Converts a number into an exponential notation*
  - *toFixed() :Formats a number with x numbers of digits after the decimal point*
  - *toPrecision(): Formats a number to x length*
  - *toString():Converts a Number object to a string*
  - *valueOf():Returns the primitive value of a Number object*

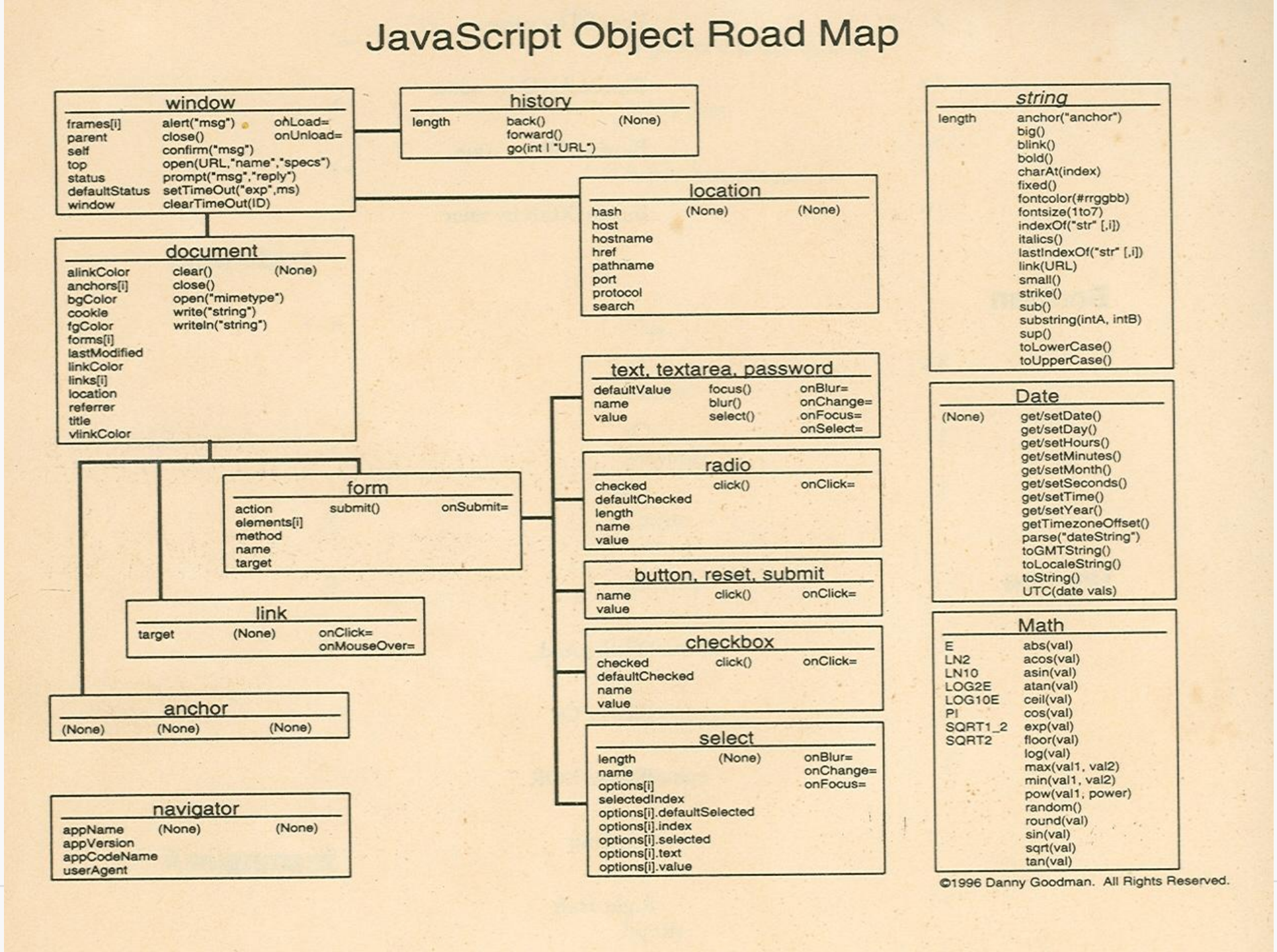
Example

JavaScript Errors

- When an error occurs, when something goes wrong, the JavaScript engine will normally stop, and generate an error message.
- The technical term for this is: JavaScript will **throw** an error.
- **JavaScript try and catch**
  - *The **try** statement allows us to define a block of code to be tested for errors while it is being executed.*
  - *The **catch** statement allows us to define a block of code to be executed, if an error occurs in the try block.*
  - *The JavaScript statements **try** and **catch** come in pairs.*
  - ***Syntax***

```
try
  {
  //Run some code here
  }
catch(err)
  {
  //Handle errors here
  }
```

Example


JavaScript Object Road Map
window
frames[i] parent self top status defaultStatus window
alert("msg") close() confirm("msg") open(URL,"name","specs") prompt("msg","reply") setTimeOut("exp",ms) clearTimeOut(ID)
onLoad= onUnload=
history
length
back() forward() go(int | "URL")
(None)
location
hash host hostname href pathname port protocol search
(None) (None)
document
alinkColor anchors[i] bgColor cookie fgColor forms[i] lastModified linkColor links[i] location referrer title vlinkColor
clear() close() open("mimetype") write("string") writeln("string")
(None)
form
action elements[i] method name target
submit()
onSubmit=
link
target
(None)
onClick= onMouseOver=
anchor
(None) (None) (None)
navigator
appName appVersion appCodeName userAgent
(None) (None)
text, textarea, password
defaultValue name value
focus() blur() select()
onBlur= onChange= onFocus= onSelect=
radio
checked defaultChecked length name value
click()
onClick=
button, reset, submit
name value
click()
onClick=
checkbox
checked defaultChecked name value
click()
onClick=
select
length name options[i] selectedIndex options[i].defaultSelected options[i].index options[i].selected options[i].text options[i].value
(None)
onBlur= onChange= onFocus=
string
length
anchor("anchor") big() blink() bold() charAt(index) fixed() fontcolor(#rrggbb) fontsize(1to7) indexOf("str" [,i]) italics() lastIndexOf("str" [,i]) link(URL) small() strike() sub() substring(intA, intB) sup() toLowerCase() toUpperCase()
Date
(None)
get/setDate() get/setDay() get/setHours() get/setMinutes() get/setMonth() get/setSeconds() get/setTime() get/setYear() getTimezoneOffset() parse("dateString") toGMTString() toLocaleString() toString() UTC(date vals)
Math
E LN2 LN10 LOG2E LOG10E PI SQRT1_2 SQRT2
abs(val) acos(val) asin(val) atan(val) ceil(val) cos(val) exp(val) floor(val) log(val) max(val1, val2) min(val1, val2) pow(val1, power) random() round(val) sin(val) sqrt(val) tan(val)
©1996 Danny Goodman. All Rights Reserved.

JavaScript Window

- The window object represents an open window in a browser.
- **Window**

Property	Description
closed	Returns a Boolean value indicating whether a window has been closed or not
defaultStatus	Sets or returns the default text in the statusbar of a window (only supported by opera)
document	Returns the Document object for the window
frames	Returns an array of all the frames (including iframes) in the current window
history	Returns the History object for the window
innerHeight	Sets or returns the inner height of a window's content area
innerWidth	Sets or returns the inner width of a window's content area
length	Returns the number of frames (including iframes) in a window
location	Returns the Location object for the window
name	Sets or returns the name of a window
navigator	Returns the Navigator object for the window

Contd..

Property	Description
opener	Returns a reference to the window that created the window
outerHeight	Sets or returns the outer height of a window, including toolbars/scrollbars
outerWidth	Sets or returns the outer width of a window, including toolbars/scrollbars
pageXOffset	Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window
pageYOffset	Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window
parent	Returns the parent window of the current window
screen	Returns the Screen object for the window
screenLeft	Returns the x coordinate of the window relative to the screen
screenTop	Returns the y coordinate of the window relative to the screen
screenX	Returns the x coordinate of the window relative to the screen
screenY	Returns the y coordinate of the window relative to the screen
self	Returns the current window
status	Sets the text in the statusbar of a window

Window Object Methods

Method	Description
alert()	Displays an alert box with a message and an OK button
clearInterval()	Clears a timer set with setInterval()
clearTimeout()	Clears a timer set with setTimeout()
close()	Closes the current window
confirm()	Displays a dialog box with a message and an OK and a Cancel button
createPopup()	Creates a pop-up window (only works in IE!)
focus()	Sets focus to the current window
moveBy()	Moves a window relative to its current position
moveTo()	Moves a window to the specified position

Contd

Method	Description
open()	Opens a new browser window
print()	Prints the content of the current window
prompt()	Displays a dialog box that prompts the visitor for input
resizeBy()	Resizes the window by the specified pixels
resizeTo()	Resizes the window to the specified width and height
scroll()	
scrollBy()	Scrolls the content by the specified number of pixels
scrollTo()	Scrolls the content to the specified coordinates
setInterval()	Calls a function or evaluates an expression at specified intervals (in milliseconds)
setTimeout()	Calls a function or evaluates an expression after a specified number of milliseconds

Open()

- **Syntax**
  - *window.open(URL,name,specs,replace)*

Parameter	Description
URL	Specifies the URL of the page to open. If no URL is specified, a new window with about:blank is opened
name	•Specifies the target attribute or the name of the window. The following values are supported: •_blank - URL is loaded into a new window. This is default •_parent - URL is loaded into the parent frame •_self - URL replaces the current page •_top - URL replaces any framesets that may be loaded •name - The name of the window
replace	•Specifies whether the URL creates a new entry or replaces the current entry in the history list. The following values are supported: •true - URL replaces the current document in the history list •false - URL creates a new entry in the history list

Contd..

specs	Optional. A comma-separated list of items. The following values are supported:	
	channelmode=yes|no|1|0	Whether or not to display the window in theater mode. Default is no. IE only
	directories=yes|no|1|0	Whether or not to add directory buttons. Default is yes. IE only
	fullscreen=yes|no|1|0	Whether or not to display the browser in full-screen mode. Default is no. A window in full-screen mode must also be in theater mode. IE only
	height=pixels	The height of the window. Min. value is 100
	left=pixels	The left position of the window
	location=yes|no|1|0	Whether or not to display the address field. Default is yes
	menubar=yes|no|1|0	Whether or not to display the menu bar. Default is yes
	resizable=yes|no|1|0	Whether or not the window is resizable. Default is yes
	scrollbars=yes|no|1|0	Whether or not to display scroll bars. Default is yes
	status=yes|no|1|0	Whether or not to add a status bar. Default is yes
	titlebar=yes|no|1|0	Whether or not to display the title bar. Ignored unless the calling application is an HTML Application or a trusted dialog box. Default is yes
	toolbar=yes|no|1|0	Whether or not to display the browser toolbar. Default is yes
	top=pixels	The top position of the window. IE only
	width=pixels	The width of the window. Min. value is 100

Open () example

Example 1

Example 2

Example 3

Window Location

- Methods to send the client to a new location (URL/page).

Example

Refresh

- How to refresh a document.

Example

Status bar

- How to write some text in the windows status bar.

Example

print a page.

- How to print the page.

Example

Browser detection

- How to detect the client's browser

Example

Browser detection & and Action

- How to detect the client's browser and take action accordingly

Example

Thank You