JAVASCRIPT

- Ms. Sonali Vaidya.

What is a scripting language?

- Simple, interpreted programming language.
- Interpreted by the browser.
- Scripts are embedded as plain text.
- Simpler execution model.
- Saves bandwidth.
- Platform – independence.
- E.g. VBScript, Jscript, Live Script, ECMA

Types of Scripting

- Client – side Scripting
  - Runs on the user's computer.
  - Source code visible to users.
  - Default Scripting
- Server – side Scripting
  - Runs on the Web server
  - Source code not visible to the user.
  - Need to specify type of scripting.

What is JavaScript???

- A light-weight programming language that runs on the Web browser (client side).
- Embedded in HTML files and manipulate the HTML itself.
- all modern web browsers support
- Interpreted, not compiled.
- Minimal Syntax
- Object Based (not Object Oriented) programming language
  - Very limited object creation.
  - Has a set of pre – defined objects associated with HTML doc. structure and browser functionality.

What is JavaScript??? …contd

- JavaScript is **<u>not</u>** Java
  - Developed by Netscape, not Sun.
  - Originally called LiveScript.
  - Netscape in alliance with Sun jointly announced the language and its new name JavaScript.
  - Because of rapid acceptance by the web community Microsoft forced to include in IE Browser

Java Vs JavaScript

Java	JavaScript
Programming Language	Scripting Language
Java is Object Oriented Programming Language	JavaScript is Object Based Programming Language
Code is Compiled	Code is Interpreted
Developed by Sun	Developed by Netscape in collaboration with Sun
Objects in Java are Static , i.e. the no. of data members and method are fixed at the compile time.	Objects in JavaScript are Dynamic, i.e. we can change the total no. of data members and method of an object during execution.

What can Javascript do???

- Gives HTML designers a programming tool.
- Put dynamic text into HTML page.
- Can react to events.
- Can read and write HTML elements.
- Can be used to validate data.
- Detect visitor’s browser.
- Used to create cookies.

Why to use JavaScript???

- To add dynamic function to your HTML
  - Adds logic to HTML.
  - Can change HTML after loading the page.

- To shoulder some form processing burden
  - Runs in the browser, not on the Web Server.
  - JavaScript can validate the data that users enter into the form, before it is sent to your Web application.

Problems with JavaScript

- If scripts doesn’t work then page is useless
- Scripts can run slowly
- Complex scripts can take a long time to start up

When not to use JavaScript???

- When you need to access other resources.
  - Files
  - Programs
  - Databases
- When you are using sensitive or copyrighted data or algorithms.
  - Your JavaScript code is open to the public.

Where does my code go?

- Code must be placed within `<script>` tags
- The `<script>` tag should include the attribute-value pair `language="javascript"`
- The contents of the **`script`** element should be enclosed in a HTML comment (i.e. within <!-- -->)
- You can use as many fragments as you want within a web page
- These can be placed in the head and body
- **`document.write`** writes to the screen
- HTML can be included in output strings

Add Script to your HTML

- In an external file

```
<html>
<head>
<script language="javascript"

    src="script.js">
</script>
</head>
```

Add Script to your HTML

- Within the HTML Page itself
  - Within the head tag

    <html><head>

    <script language="javascript">

    // JavaScript code

    </script>
  - Within the html body
  - Within some HTML Tags as Event Handlers.

Integrating Script under the <head> tag

- In the head section it will execute when called i.e. JavaScript in the HTML will execute immediately while the web page loads into the web browser before anyone uses it.

```
<html>
<head>
<title> My First JavaScript </title>
<script type= “text/javascript”>
Some statements
</script>
</head>
<body>
</body>
</html>
```

External Script

- We can write the code we want in a separate file out of our program and save it with the extension .js.
- The script tag is not needed in the external file, just the code.
- The JavaScript must then be linked to the point where we want it to be inserted in our HTML file.
- If the script is saved as myfile.js, then we call this script , using the src attribute, from any of our pages.

```
<html>
<head>
<title> My First JavaScript </title>
</head>
<body>
<script src="myfile.js">
</script>
</body>
</html>
```

Integrating Script under the <body> tag

- When we place JS code under the <body> tag; this generates the content of the web page.
- JS code under the <body> tag executes when the page loads & go in the body section.

```
<html>
<head>
<title> My First JavaScript </title>
</head>
<body>
<script type= “text/javascript”>
Some statements
</script>
</body>
</html>
```

Sample Program

```
<html>
<head>
<title> My First JavaScript </title>
</head>
<body>
<script type= “text/javascript”>
document.write (“ My First Javascript Program.”);
</script>
</body>
</html>
```

- document.write statement will display the desired message on the web browser.
- Statements ends with semicolon.
- (In JavaScript semicolon indicates the end of the Statement.)
- If you want to write the block of statements then write it between curly brackets.

```
<html>
<head>
<title> My First JavaScript </title>
</head>
<body>
<script type= “text/javascript”>
{
document.write (“ <h1>This is Header </h1>”);
document.write (“ <p> This is Paragraph. </p>”);
}
</script>
</body>
</html>
```

Comments

- Single Line comments will start with //
- Multiple Line Comments will
  start with /*
  end with */

Browsers without JS support

- Not all browser support JavaScript.
- Browsers without JS support do not recognize the start & end script tag. Therefore they ignore tags.
- As a result they will display it as a text document.
- So to keep browser without JS support as it is from displaying the statements in a script on screen, enclose the script's statement with HTML start & end comment tags.

```
<html>
<head>
<title> My First JavaScript </title>
</head>
<body>
<script language= “javascript”>
< !--
document.write (“ <h1>This is Header </h1>”);
// -->
</script>
</body>
</html>
```

- Browser that do not support JS will ignore the start & end script tag & everything between HTML start & end tags .
- Conversely , JS aware browser ignore the HTML comment tags when the tags occur within a set of start & end script tags.

Variables

- JS allow you to store information in named variables, which are just storage memories so we can access the data later by using a name which we have assigned.
- A variable's value can change as the script runs too.
- A *Variable* can contain several types of value.

number	A numeric value
string	Character wrapped in quotes
boolean	A value of true or false
null	An empty variable
function	A function
object	An object

- These are known as *primitive data type*, with only a single value. JS also supports *composite data type* known as an *object* .

JS variables name has few rules:-

- It is case sensitive.
- Variable name should to start with alphabet or underscore while rest of the name can be numeric, dollar sign and underscore character.
- It can not contain punctuation, spaces or start with a digit.
- It can not be a JS reserved word.

Variables

- Javascript has unsigned variables.
- Variables are declared with the **var** keyword:
  - var num=“2”;
  - var nam=“ABCD”;
  - var ph_no=“022-28928585”;

Variable Scope

- Variables defined *outside* functions are *global functions*.
- Variables defined *inside* functions are *local functions*.
- Scope of global variable limited to the current document.
- *"var"* usually optional for global variables.
- JS contains functions that can be called from anywhere in the main program.
- The instant the page is unloaded, all global variables are erased from the memory.
- If var is not at all used then the variable will have a global scope whether in a local function or not.

Declaring (Creating) JavaScript Variables

- Creating a variable in JavaScript is most often referred to as "declaring" a variable.
- You declare JavaScript variables with the **var** keyword :

```
var student_name;
```

- After the declaration, the variable is empty (it has no value).
- To assign a value to the variable, use the equal sign:

```
student_name=“ABC";
```

- However, you can also assign a value to the variable when you declare it:

```
var student_name="ABC";
```

- In the example below we create a variable called student_name, assigns the value "ABC" to it,

- You can declare many variables in one statement. Just start the statement with **var** and separate the variables by comma:

```
var student_name="ABC", age=20,department="INFT";
```

```
<html >
<head>
<title>javascript variables</title> </head>
 <body>
 <script type="text/javascript">
var bookname;
var bookprice;
bookname="Javascript fundamentals";
bookprice=500;
document.write("Name of the book is : ",bookname);
document.write("<br>");
document.write("Price of the book is : ",bookprice);
</script>
</body>
</html>
```

Output :-

```
Name of the book is : Javascript fundamentals
Price of the book is : 500
```

Functions

- A function is a reusable block of statements that perform particular task.
- In JS a function encapsulates in a Block of statement.
- Syntax of function :-

```
function function_name (parameter)
{
        Statements;
}
```

- The function is defined using the function keyword followed by the name of the function.
- The function should be meaningful.

- The block of statements inside curly brackets fulfill the purpose of the function.
- The function gets its value through the parameter.
- A function contains the code that will be executed by an event or call to that function.
- We can call the function from anywhere within the program.
- We can define the function in the head and body section of the html.
- When we call a function, we simply write its name followed by a parenthesis.
- We can also use parenthesis to pass the parameter to the function in which we can define the value of the parameter.

```
<html>
<head>
<title>funtions</title>
<script type="text/javascript">
function my_fun()
{
document.write("This Statement is within the function.");
}
</script>
</head>
<body>
<p> functions </p>
<script type="text/javascript">
document.write("This Statement is before a function call.");
document.write("<br>");
my_fun();
</script>
</body>
</html>
```

Output:-

```
Functions

This Statement is before a function call.
This Statement is within the function.
```

Lecture 2

Objects in JavaScript

- In Javascript we can create objects in two ways
  - By creating a direct instance

OR

  - By creating an object using a function template

Creating Direct instance of an object

- Direct instance of an object is created by using **new** keyword

```
Obj= new object();
```

- We can add properties and methods to an object by using a period (.) followed by a property or method name.
- Obj.name="ABC"
- Obj.rollnumber=32
- Obj.getValue()

Obj is a newly created object while name and rollnumber are the properties and getValue is a method

Creating a Function template

- After creating function template of an object , we need to create an instance of an object by using the **`new`** keyword
- A function template for an object is created by using a **`function`** keyword.
- We can also add properties to function by using the **`this`** keyword
- **Note** -

Creating a Function template

```
function book(bookname,author,price)
{
this.bookname=bookname;
this.author=author;
this.price=price;
}
```

- Here function named book is created and three parameters are added to it.
- We can assign values to the parameters.
- this keyword is used to represent the current object that is in use.

Creating a Function template (Continue)

- Next we have to create Create the instance of the object book by using the new keyword

```
var my_book=new book("ABC","XYZ",100);
```

- We can access the my_book instance of the book object,

```
var book_type= my_book.price;
```

Method of an object

- Method can be defined as action performed by an object
- A complex a application is divided into methods; as a result code becomes more flexible, easy to maintain and easy to debug.
- Methods also increase the reusability of a code as we can execute a code for a n number of times by just calling method name.
- To add methods to a user-defined object, we need to perform following steps:-
- Declaring and defining a function for each method
- Associating a function with an object

Method of an object

- Assume a triangle as an object and create a function calarea()

```
//method function
 function calarea()
{
var area=this.base*this.altitude*0.5;
Return area;
}
```

- Associate the function calarea() to the object traingle

```
<script type="text/javascript">
Function triangle(b,a)
{
this.base=b;
this.altitude=a;
this.area=calarea
}
</script>
```

- call the method by instantiating the object,

```
<script type="text/javascript">
Var mytriangle=new triangle(20,10)
Alert("area="+mytriangle.area())
</script>
```

JavaScript EventHandlers

- onAbort – activated when a user aborts the loading of an image
- onBlur - Used with frame, select, text, textarea and window objects
  - invoked when an object loses the focus
- onChange - Used with select, text and textarea objects
  - use instead of onBlur to validate only if a value has changed
- onClick - Used with button, checkbox, link, radio, reset, and submit objects.

JavaScript Event Handlers

- onError - Used with image and window objects to invoke a handler if an error occurs while an image or window is loading.
- onFocus - Used with frame, select, text, textarea and window objects.
  - Just the opposite of onBlur; i.e. invoked when the object gets focus.
- onLoad - Used with window, frame and image objects (use with <body ….><frameset ….> and <img ...>)

JavaScript Event Handlers

- onMouseOut and onMouseOver
- onReset – used with form objects.
- onSelect - Used with text and textarea objects
  - run some JavaScript whenever a user selects a piece of text in a text or textarea object.
- onSubmit - Use with form objects to run a handler whenever a form has been submitted.
  - Useful to validate all fields prior to actual submission

Event Example

```
<html>
    <head>
        <script language=“javascript”>
            function funct() {
                // code
            }
        </script>
    </head>
    <body>
        <input type=“Button” name=“Click” …
    onClick=“funct();”>
    </body>
</html>
```

Lecture 3

Operators

- Arithmetic Operators
  - Standard math operators like +, -, *, / are built into JavaScript.
- Comparison Operators
  - Return true or false
  - ==, !=, >, >=, <, <=
- Boolean Operators
  - &&, | |, !

Operators

- Bitwise operators.
  - Applied to each corresponding pair of bits from both the operands.
  - &, |, <<, >> (binary), ~ (unary) operators.

- Assignment and Aggregate operators
  - = is an assignment operator.
  - +=, -=, *=, /= are aggregate operators.

Dialog Boxes and Prompts

- Alerts – display message in a small window and wait for the user to respond by clicking the OK button.

```
alert('Message');
```

- Prompt – displays a dialog box with provision for data entry along with two buttons OK and Cancel.

```
prompt("Your Message","defaultMsg");
```

Dialog Boxes and Prompts

- confirm – a dialog box along with OK and Cancel buttons.

```
confirm (“Are you sure???”);
```

Control Structures

- if construct
- if … else construct
- for loops
- do … while loops
- while loops
- switch … case
- break
- continue

if statement:

- The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.
- **Syntax:**

```
if (expression)
{ Statement(s) to be executed if expression is true }
```

- **if...else statement:**
- The **if...else** statement is the next form of control statement that allows JavaScript to execute statements in more controlled way.
- **Syntax:**

```
if (expression)
{ Statement(s) to be executed if expression is true }
Else
{ Statement(s) to be executed if expression is false }
```

if...else if... statement:

- The **if...else if...** statement is the one level advance form of control statement that allows JavaScript to make correct decision out of several conditions.
- **Syntax:**

```
if (expression 1)
{ Statement(s) to be executed if expression 1 is true }
else if (expression 2)
{ Statement(s) to be executed if expression 2 is true }
else if (expression 3)
{ Statement(s) to be executed if expression 3 is true }
Else
{ Statement(s) to be executed if no expression is true }
```

switch statement:

- The basic syntax of the **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.
- Syntax :

```
switch (expression)
 { case condition 1: statement(s)
                     break;
   case condition 2: statement(s)
                     break;
   ...
   case condition n: statement(s)
                     break;
  default: statement(s)
 }
```

The *while* Loop

- The most basic loop in JavaScript is the **while** loop which would be discussed in this tutorial.
- **Syntax:**

```
while (expression)
{
Statement(s) to be executed if expression is true
}
```

The *do...while* Loop:

- The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is *false*.
- **Syntax:**

```
do
{
Statement(s) to be executed;
}
 while (expression);
```

The *for* Loop

- The **for** loop is the most compact form of looping and includes the following three important parts:
- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out.
- The iteration statement where you can increase or decrease your counter.
- You can put all the three parts in a single line separated by a semicolon.

- **Syntax:**

```
for (initialization; test condition; iteration statement)
{
Statement(s) to be executed if test condition is true
 }
```

- **The *for…in* Loop**
- Syntax:

```
for (variablename in object)
{ statement or block to execute }
```

- In each iteration one property from *object* is assigned to *variablename* and this loop continues till all the properties of the object are exhausted.

The *break* & *continue* Statement:

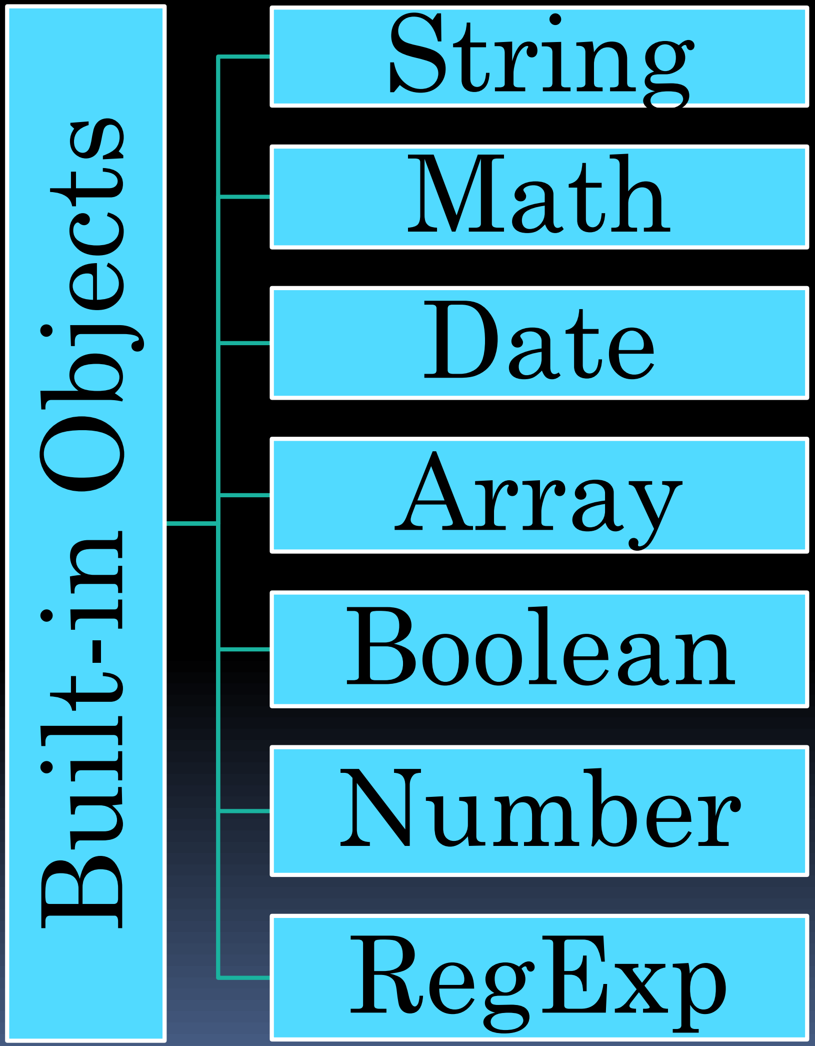
- The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

- The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip remaining code block.

- When a **continue** statement is encountered, program flow will move to the loop check expression immediately and if condition remain true then it start next iteration otherwise control comes out of the loop.

JavaScript's Built-in Object

Built-in Object in JavaScript



The String Object

- The **String** object let's you work with a series of characters and wraps Javascript's string primitive data type with a number of helper methods.
- Because Javascript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.
- **Syntax:**
- Creating a **String** object:

```
var val = new String(string);
```

- The *string* parameter is series of characters that has been properly encoded.

String Properties:

Property	Description
constructor	Returns a reference to the String function that created the object.
length	Returns the length of the string.
prototype	The prototype property allows you to add properties and methods to an object.

Method	Description
charAt()	Returns the character at the specified index.
charCodeAt()	Returns a number indicating the Unicode value of the character at the given index.
concat()	Combines the text of two strings and returns a new string.
indexOf()	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
lastIndexOf()	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
localeCompare()	Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
match()	Used to match a regular expression against a string.
replace()	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.

Method	Description
search()	Executes the search for a match between a regular expression and a specified string.
slice()	Extracts a section of a string and returns a new string.
split()	Splits a String object into an array of strings by separating the string into substrings.
substr()	Returns the characters in a string beginning at the specified location through the specified number of characters.
substring()	Returns the characters in a string between two indexes into the string.
toLocaleLowerCase()	The characters within a string are converted to lower case while respecting the current locale.
toLocaleUpperCase()	The characters within a string are converted to upper case while respecting the current locale.
toLowerCase()	Returns the calling string value converted to lower case.
toString()	Returns a string representing the specified object.
toUpperCase()	Returns the calling string value converted to uppercase.
valueOf()	Returns the primitive value of the specified object

- **Object: String**
  Method or Function: length
  Description: This will count the total number of characters (length or size) present in the string and returns the value.

  Example Code:

  ```
  <script language=javascript>
  var ss = "a character count - size test ";
  var result = ss.length;
  document.write(result);
  </script>
  ```

  Result: 30

- **Explanation:**
  In the above example , the string is stored in a variable ss. Then the function is called as "ss.length". the method counts the total characters in the string and returns it. the result is of variable type, integer.

- **Object**: **String**
  Method or Function: **toLowerCase(), toUpperCase()**
  Description: These methods are used to cover a string or alphabet from lower case to upper case or vice versa. e.g: "and" to "AND".

  Converting to Upper Case:
  Example Code:
```
<script language=javascript>
var ss = " testing case conversion method ";
var result = ss.toUpperCase();
document.write(result);
</script>
```

  Result: TESTING CASE CONVERSION METHOD

- **Converting to Lower Case:**
  Example Code:
```
<script language=javascript>
var ss = " TESTING LOWERCASE CONVERT FUNCTION ";
var result = ss.toLowerCase();
document.write(result);
</script>
```

  Result: testing lowercase convert function

Math Object

- The **math** object provides you properties and methods for mathematical constants and functions.
- Unlike the other global objects, *Math* is not a constructor. All properties and methods of Math are static and can be called by using *Math* as an object without creating it.
- Thus, you refer to the constant pi as **Math.PI** and you call the *sine* function as **Math.sin(x)**, where x is the method's argument.
- Syntax:

```
var pi_val = Math.PI;
var sine_val = Math.sin(30);
```

- Here is the simple syntax to call properties and methods of Math.

Math Properties:

Property	Description
E	Euler's constant and the base of natural logarithms, approximately 2.718.
LN2	Natural logarithm of 2, approximately 0.693.
LN10	Natural logarithm of 10, approximately 2.302.
LOG2E	Base 2 logarithm of E, approximately 1.442.
LOG10E	Base 10 logarithm of E, approximately 0.434.
PI	Ratio of the circumference of a circle to its diameter, approximately 3.14159.
SQRT1_2	Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.
SQRT2	Square root of 2, approximately 1.414.

Math Methods

Method	Description
abs()	Returns the absolute value of a number.
acos()	Returns the arccosine (in radians) of a number.
asin()	Returns the arcsine (in radians) of a number.
atan()	Returns the arctangent (in radians) of a number.
atan2()	Returns the arctangent of the quotient of its arguments.
ceil()	Returns the smallest integer greater than or equal to a number.
cos()	Returns the cosine of a number.
exp()	Returns E^N, where N is the argument, and E is Euler's constant, the base of the natural logarithm.
floor()	Returns the largest integer less than or equal to a number.

Math Methods

Method	Description
log()	Returns the natural logarithm (base E) of a number.
max()	Returns the largest of zero or more numbers.
min()	Returns the smallest of zero or more numbers.
pow()	Returns base to the exponent power, that is, base exponent.
random()	Returns a pseudo-random number between 0 and 1.
round()	Returns the value of a number rounded to the nearest integer.
sin()	Returns the sine of a number.
sqrt()	Returns the square root of a number.
tan()	Returns the tangent of a number.
toSource()	Returns the string "Math".

Javascript Math - sqrt Method

- **Description:**
- This method returns the square root of a number. If the value of number is negative, sqrt returns NaN.
- **Syntax:**

```
Math.sqrt( x ) ;
```

- Here is the detail of parameters:
- **x**: A number
- **Return Value:** Returns the square root of a number.

```
<html>
<head>
<title>JavaScript Math sqrt() Method</title>
 </head>
 <body>
<script type="text/javascript">
var value = Math.sqrt( 81 );
document.write("<br />First Test Value : " + value );
var value = Math.sqrt( -4 );
document.write("<br /> Second Test Value : " + value );
</script>
</body>
 </html>
```

- This will produce following result:

```
First Test Value : 9
Second Test Value : NaN
```

The Date Object

- Date is a predefined object in javascript.
- This object contains methods that can be used to get or set the system date, time, month, day, hours, minutes, seconds, etc…. Contains methods to get and set properties of UTC universal coordinated time. Universal Coordinated Time, UTC is the world time standard also called as Greenwich Mean Time.
- A new object has to be created before using its methods.
- **Creation of new date object:**
- Example Code: **var exd = new Date();**
- Calling "new Date()" will create a new date object. This object can be used to execute its methods or functions. We have to capture the object in a variable.

  In the example the object it stored in a variable name exd.

Method	Example Code	Result	Description

getDate()	exd = new Date();
	exd.getDate();
		17	getDate() method returns today's date of the month as in your computer. The range is from 1 to 31.

getUTCDate()	exd = new Date();
	exd.getUTCDate();
		17	getUTCDate() method return the date value of Universal Coordinated Time. UTC is the world time standard also called as Greenwich Mean Time.

getMonth()	exd = new Date();
	exd.getMonth();
		2	getMonth() method returns the Month of this year, as in your computer. The result can range from 0 to 11. 0 for january to 11 for december.

getUTCMonth()	exd = new Date();
	exd.getUTCMonth();
		2	getUTCMonth() returns the UTC month of the year, as in your computer.

getDay()	exd = new Date();
	exd.getDay();
		0	getDay() method returns the day of the week, as in your computer. The result can range from 0 to 6. 0 for Sunday, 1 for Monday, up to 6 for Saturday.

getUTCDay()	exd = new Date();
	exd.getUTCDay();
		0	getUTCDay() returns the UTC day of the week, as in your computer.

getHours()	exd = new Date();
	exd.getHours();
		23	getHours() method returns the current hour of the day, as in your computer. The result can range is from 0 to 24.

getUTCHours()	exd = new Date();
	exd.getUTCHours();
		18	getUTCHours() returns the UTC hours of the day, as in your computer. **[Indian Standard Time is 5.30 hours ahead of UTC]**.

getMinutes()	exd = new Date();
	exd.getMinutes();
		59	getMinutes() method returns the minutes of this hour, as in your computer. The result can range is from 0 to 59.

getUTCMinutes()	exd = new Date();
	exd. getUTCMinutes();
		29	getUTCMinutes() returns the UTC minutes of the hour, as in your computer.

Method	Example Code	Result	Description
getSeconds()	exd = new Date(); exd.getSeconds();	18	getSeconds() method returns the Seconds of this minute, as in your computer. The result can range is from 0 to 59.
getUTCSeconds ()	exd = new Date(); exd. getUTCSeconds();	18	getUTCSeconds() returns the UTC seconds of the minute, as in your computer.
getMilliseconds ()	exd = new Date(); exd.getMilliseconds() ;	467	getMilliseconds() method returns the milli seconds of current second, as in your computer.
getTime()	exd = new Date(); exd.getTime();	-	Result: 1363544958467 getTime() method returns the current time since 1/1/1970 in milli seconds, as in your computer.
getYear()	exd = new Date(); exd.getYear();	113	getYear() method returns the year since 1900, as in your computer.
getFullYear()	exd = new Date(); exd.getFullYear();	2013	getFullYear() method returns the full year, as in your computer.
toGMTString()	exd = new Date(); exd.toGMTString();	Sun, 17 Mar 2013 18:29:18 GMT	toGMTString() method returns the date and time in Internet Greenwich Mean Time Format (GMT).

The Arrays Object

- The **Array** object let's you store multiple values in a single variable.
- Syntax for Creating a **Array** object:

```
var fruits = new Array( "apple", "orange", "mango" );
```

- The *Array* parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.
- Y g values as follows:

```
var fruits = [ "apple", "orange", "mango" ];
```

- You will use ordinal numbers to access and to set values inside an array as follows:

```
fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element
```

Array Properties:

Property	**Description**
constructor	Returns a reference to the array function that created the object.
index	The property represents the zero-based index of the match in the string
input	This property is only present in arrays created by regular expression matches.
length	Reflects the number of elements in an array.
prototype	The prototype property allows you to add properties and methods to an object.

Array Methods

Method	Description
concat()	Returns a new array comprised of this array joined with other array(s) and/or value(s).
every()	Returns true if every element in this array satisfies the provided testing function.
filter()	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
forEach()	Calls a function for each element in the array.
indexOf()	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
join()	Joins all elements of an array into a string.
lastIndexOf()	Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
map()	Creates a new array with the results of calling a provided function on every element in this array.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of an array and returns the new length of the array.

Method	Description
reduce()	Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
reduceRight()	Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
shift()	Removes the first element from an array and returns that element.
slice()	Extracts a section of an array and returns a new array.
some()	Returns true if at least one element in this array satisfies the provided testing function.
toSource()	Represents the source code of an object
sort()	Sorts the elements of an array.
splice()	Adds and/or removes elements from an array.
toString()	Returns a string representing the array and its elements.
unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

Javascript Array concat() Method

- Javascript array **concat()** method returns a new array comprised of this array joined with two or more arrays.
- **Syntax:**

```
array.concat(value1, value2, ..., valueN);
```

- Here is the detail of parameters:
- **valueN** : Arrays and/or values to concatenate to the resulting array.
- **Return Value:** Returns the length of the array.

```
<html>
<head>
<title>JavaScript Array concat Method</title>
 </head>
 <body>
<script type="text/javascript">
 var alpha = ["a", "b", "c"];
var numeric = [1, 2, 3];
var alphaNumeric = alpha.concat(numeric);
document.write("alphaNumeric : " + alphaNumeric );
</script>
 </body>
</htm>
```

- This will produce following result:

```
alphaNumeric : a,b,c,1,2,3
```

JavaScript Regular Expression

- A regular expression is an object that describes a pattern of characters.
- When you search in a text, you can use a pattern to describe what you are searching for.
- A simple pattern can be one single character.
- A more complicated pattern can consist of more characters, and can be used for parsing, format checking, substitution and more.
- The JavaScript **RegExp** class represents regular expressions, and both String and **RegExp** define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

- Syntax:

```
var pattern = new RegExp(pattern, attributes);
or simply
var pattern = /pattern/attributes;
```

Here is the description of the parameters:

- ***pattern:*** A string that specifies the pattern of the regular expression or another regular expression.

- ***attributes:*** An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multiline matches, respectively.

Brackets:

- Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Expression	Description
[...]	Any one character between the brackets.
[^...]	Any one character not between the brackets.
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

Quantifiers:

- The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, *, ?, and $ flags all follow a character sequence.

Expression	Description
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing one or more p's.
p{**N**}	It matches any string containing a sequence of **N** p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.

Modifiers

- Several modifiers are available that can make your work with regexps much easier, like case sensitivity, searching in multiple lines etc.

Modifier	Description
i	Perform case-insensitive matching.
m	Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of a string boundary
g	Perform a global match that is, find all matches rather than stopping after the first match.

Example

Do a case-insensitive search for “abc" in a string:

```
<html>
<body>
<script>
var str = "hiii ABC";
var patt1 = /abc/i;
document.write(str.match(patt1));
</script>
</body>
</html>
```

OUTPUT :- ABC

- **Do a global search for "is":**

```
<html>
<body>
<script>
var str="Is this all there is?";
var patt1=/is/g;
document.write(str.match(patt1));
</script>
</body>
</html>
```

- OUTPUT :- is,is

- **Do a global, case-insensitive search for "is":**

```
<html>
<body>
<script>
var str="Is this all there is?";
var patt1=/is/gi;
document.write(str.match(patt1));
</script>
</body>
</html>
```

- OUTPUT :- Is,is,is

RegExp Properties:

Property	Description
constructor	Specifies the function that creates an object's prototype.
global	Specifies if the "g" modifier is set.
ignoreCase	Specifies if the "i" modifier is set.
lastIndex	The index at which to start the next match.
multiline	Specifies if the "m" modifier is set.
source	The text of the pattern.

RegExp Methods:

Method	Description
exec()	Executes a search for a match in its string parameter.
test()	Tests for a match in its string parameter.
toSource()	Returns an object literal representing the specified object; you can use this value to create a new object.
toString()	Returns a string representing the specified object.

The test() method

- It searches a string for a specified value, and returns true or false, depending on the result.
- The following example searches a string for the character "e":

```
<html>
<body>
<script>
var patt1=new RegExp("e");
document.write(patt1.test("The best things in life are free"));
</script>
</body>
</html>
```

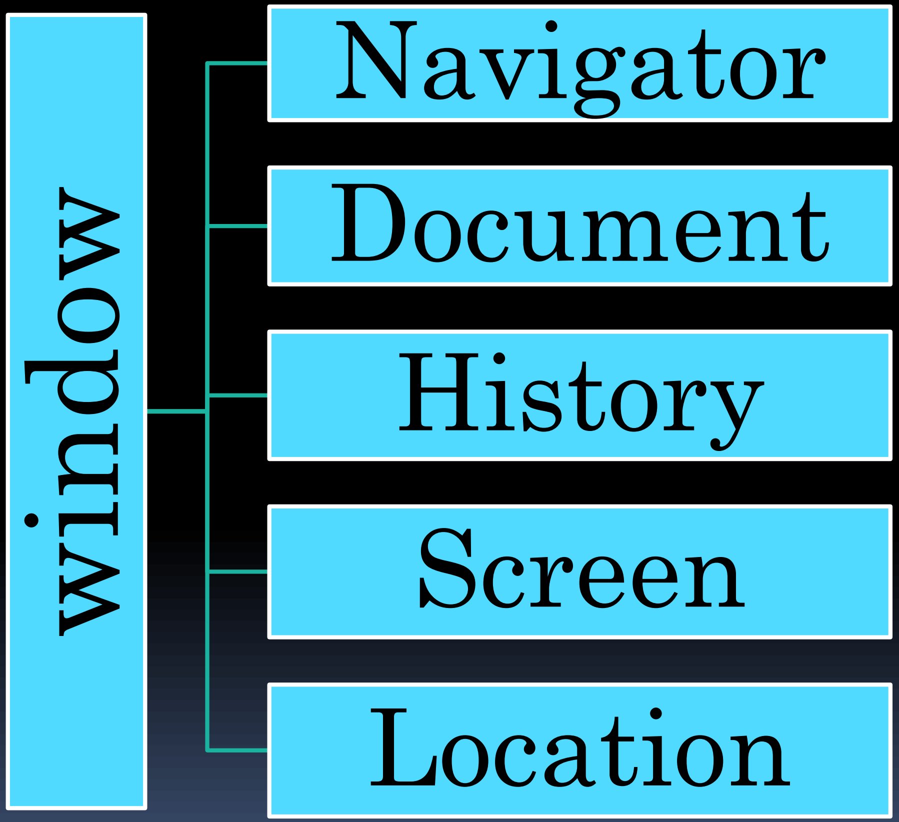
```
OUTPUT - true
```

The exec() method

- It searches a string for a specified value, and returns the text of the found value. If no match is found, it returns *null*.
- The following example searches a string for the character "e":

```
<html>
<body>
<script>
var patt1=new RegExp("e");
document.write(patt1.exec("The best things in life are free"));
</script>
</body>
</html>
OUTPUT - e
```

Browser Objects in JavaScript



Window Object

- The top level object in the JavaScript hierarchy.
- It corresponds to the current web browser window which may contain a page or no of pages in frames.
- All kinds of dialog boxes can be displayed with it.
- A Window object is created automatically with every instance of a <body> or <frameset> tag

The Window Object … contd

- Accessing window properties
  - window.*propertyName*
  - window.*methodName([parameters])*

Window properties and methods

- Methods
  - alert(message)
  - setInterval() &clearInterval()
  - setTimeOut() &clearTimeOut()
  - close()
  - confirm(message)
  - prompt(message, defaultReply)

Location Object

- Location object is a property of the window object.
- Refers to information about the URL of any currently open window or of a specific frame.
- It is part of the Window object.
- and is accessed through the window.location property.

Document object

- Document object is part of window object
- The Document object represents the entire HTML document.
- and can be used to access all elements in a page.
- is accessed through the window.document property.

History object

- The history corresponds to the ordered list of URLs.
- that user has been to before the current page.
- The History object is part of the Window object.
- and is accessed through the window.history property.

Navigator /Screen Object

- The Navigator object is automatically created by the JavaScript runtime engine.
- It contains information about the client browser.
- Screen object contains information about the client's display screen.

Functions

- JavaScript instructions are usually grouped together in a ***function***:
    - <script language=“javascript”>
        - **function *myFunction*(*parameters*) {**
            - **// some logical grouping of code**
        - **}**
    - </script>
- Like a method, procedure, or subroutine.
- Functions are called by ***events***.

Events

- JavaScript is **event-driven**.
  - Something has to happen before the JavaScript is executed.
- Each object has particular events that they will respond to.
- The way you specify an event handler is by adding an additional attribute to the HTML tag that specifies the particular handler.

Cookies

- a cookie is a small file containing information, which a server embeds on a user's computer .
- Each time the same computer requests for a web page from a server , the server refers to the created cookie for displaying the requested web page.
- Size of the cookie is depends on a browser.
- It should not exceed 1K (1024 bytes) .
- A cookie is generally used to store the username and password information on a computer so that no need to enter this information each time you visit a website.

Javascript Cookies

Attribute	Description
Name	Represents a variable name and the corresponding value to be stored in a cookie
Expires	Specifies the time when a cookies is deleted
Domain	Represents a domain name (partial or complete) to which a cookie is sent.
Path	Identifies sites among various paths in the same domain. Setting this attribute to the server root(/) allows the entire domain to access the information a stored in a cookie
secure	Restricts a browser from sending cookie information over unsecured connections. It allows the cookie to be sent over any type of HTTP connection. The default value of the Security Flag attributes is 0. Setting the value to 1 allows the cookie to be sent over a secure HTTP connection.

- JavaScript can also manipulate cookies using the *cookie* property of the *Document* object. JavaScript can read, create, modify, and delete the cookie or cookies that apply to the current web page.

Storing Cookies:

- The simplest way to create a cookie is to assign a string value to the *document.cookie* object, which looks like this:
- **Syntax:**

```
document.cookie = "key1=value1;key2=value2;expires=date";
```

- Here *expires* attribute is option. If you provide this attribute with a valid date or time then cookie will expire at the given date or time and after that cookies' value will not be accessible.

- **Note:** Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript *escape()* function to encode the *value* before storing it in the cookie. If you do this, you will also have to use the corresponding *unescape()* function when you read the cookie value.

Reading Cookies:

- Reading a cookie is just as simple as writing one, because the value of the *document.cookie* object is the cookie. So you can use this string whenever you want to access the cookie.

- The *document.cookie* string will keep a list of *name=value* pairs separated by semicolons, where*name* is the *name* of a cookie and value is its string value.

- You can use strings' *split()* function to break the string into key and values as follows:

Setting the Cookies Expiration Date:

- You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiration date within the cookie.
- This can be done by setting the *expires* attribute to a date and time.

- **Example:**

  The following example illustrates how to set cookie expiration date after 1 Month :

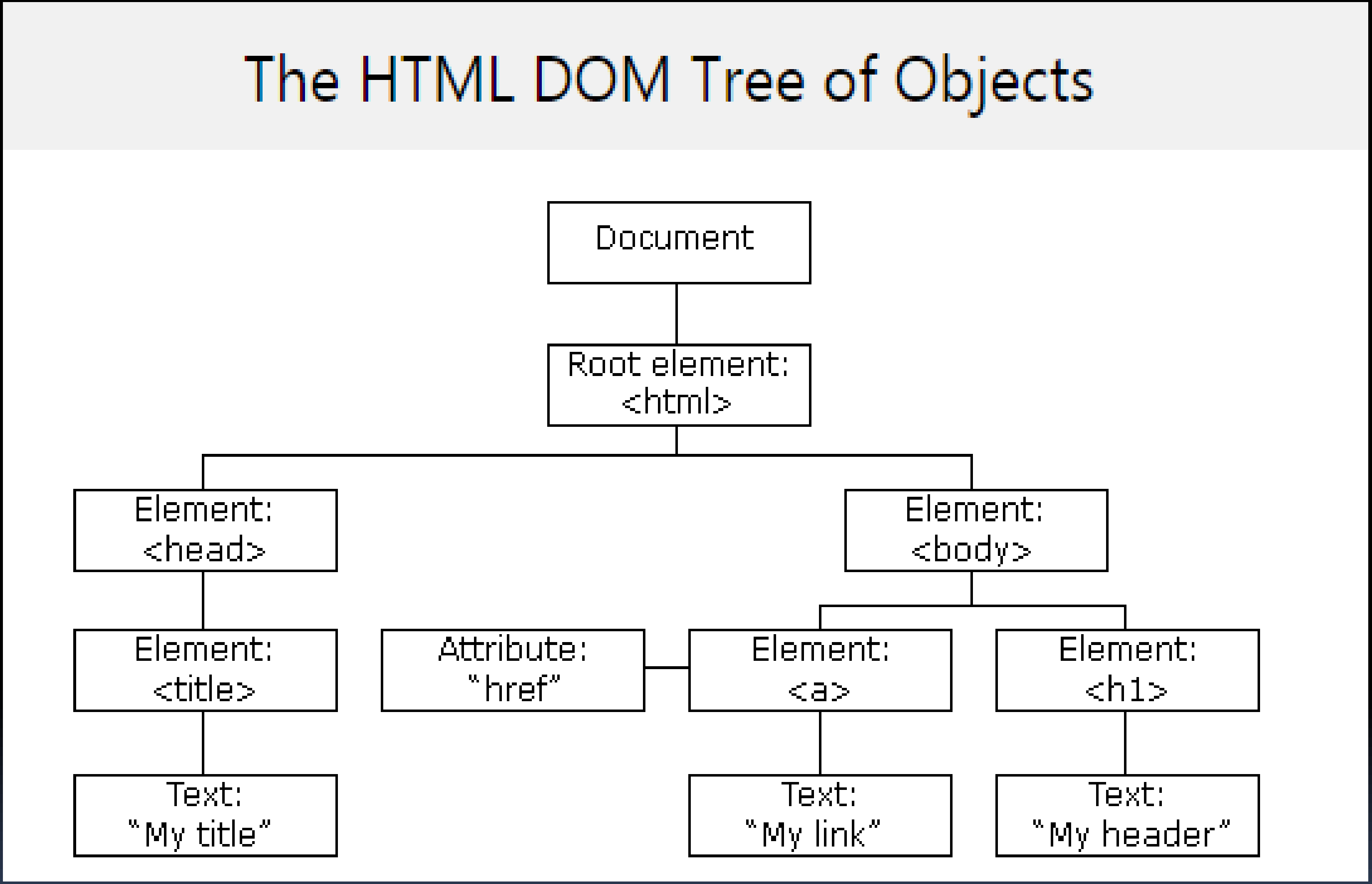
Deleting a Cookie:

- Sometimes you will want to delete a cookie so that subsequent attempts to read the cookie return nothing.
- To do this, you just need to set the expiration date to a time in the past.

- **Example:**

  The following example illustrates how to delete cookie by setting expiration date one Month in past :

DOM



The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

Thank You ☺