Javascript

We have HTML to build a web page and CSS for decoration. But both these languages cant add dynamic behavior to a web page.

So what is dynamic behavior?

Dynamic behavior means

executing conditional statements

responding to user behavior like clicks,

validating forms

getting data from the server

changing the html view depending upon conditions or user response.

Javascript is the tool which is used to add dynamic effects to the web page.

What is JavaScript

JavaScript was originally created by Netscape as a means to add a light programming language to run on web browsers. Soon JavaScript gained popularity and all major browsers added support to JavaScript

JavaScript is actually implementation of ECMAScript. A standard set by ECMA International Society.

The main feature of JavaScript is its ability to interact with HTML via DOM, application programming interface to interact with HTML.

Variables in Javascript

Javascript variables are defined using var keyword.

Ex: var name = “AcadGild”;

Variable names are case sensitive

They must begin with a letter or the underscore character.

JavaScript is untyped programming language.

This means a JavaScript variable can hold a value of any data type. We don't have to tell JavaScript during variable declaration what type of value will be stored in variable. The value type of a variable can change during the execution of program is taken care automatically.

Semicolons in javascript are terminators of statements.line can also be terminated by giving a line break. Note that though semcolon in javascript are optional, it is a recommended practice to use semicolons.

Writing firstJavascript program

Lets write a small javascript program. Open any editor of your choice and write the following lines of code and save the file as .html file. Following are the points to note:

1Javascript is written in html file inside a script tag.

2)alert is a function to display popup message to user. The plus (+) operator is used to add strings.

- <html>
- head>
- <script type = “text/javscript”>

```
Var name  = “Smith”;

Var age = 29;

alert(“The name is “+name +”And age is “+age);

</script>

</head>

<body>

</body>

</html>
```

Data Types in JavaScript

Javascript has three primary data types, two composite data types, and two special data types . All different types are derived from the basic three data types.

Following are the Primary Data Types:

1)String; We use the string data type to represent text.

2)Number :Integer can be positive whole numbers, negative whole numbers, and 0 and can be represented in base 10 as decimal, base 16 as hexadecimal, and base 8 as octal .There is no difference between integer and float, JavaScript represents all numbers as floating-point values

3)Boolean: Boolean data type has only 2 values, True and False.

The following are 2 composite data types

1)Object :objects is a data type with collection of properties . Property values can be values of any type, including other objects, which enables building complex data structures. Properties are identified using key values

2)Array :Array are Objects to represent lists or sets . They have additional functions like push or indexOf to make data operations easier.

Special data types are

1)Null: **null** contains no valid Number, String, Boolean, Array, or Object. It is used to erase the contents of a variable without deleting the variable.

2)Undefined: A variable which is declared, but never had a value assigned get an undefined value.

We can check to see if a variable exists by comparing it to undefined.

Operators in Javascript

Javascript have operators like any other language like C, C++.

+ operators is for addition

- operators is for subtraction

-- is for Decrement operator

++ is for Increment operator

/ is for division

* is for Multiplication

% is Modulus operator

== checks for equality of 2 operands

!= is for inequality

> and < are for greater and less than respectively

>= and <= are for greater or equal and later is for less or equal

&& is logical AND

|| is logical OR

! is Logical Not

Conditional statements in Javascript

If else allows JavaScript to make decisions and execute statements conditionally.

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

Switch statement

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

break syntax indicate the end of the particular case. If omitted, the programmer will continue executing each statement in each of the cases . if nothing matches, statements within default are executed.

While and for loop

while (expression){

Statement(s) to be executed if expression is true

}

As long as expression is true, the loop will continue executing.

For Loop :

for (initialization; test condition; statement for every iteration){

Statement(s) to be executed if test condition is true

}

The example below will print Hello 5 times.

Ex:

```
var count;
for (count=0; count<5; count++){
 document.write("Hello");
}
```

Note: break can be used to exit a loop iteration. continue can be used to skip and move to the next iteration.

for in loop

Javascript has a special loop called for in loop to iterate through object's properties. We will discuss about objects in depth later.

For now lets understand that we can create object by var student = {};

Student.name adds a name property to the student object. In the loop given, we iterate through All the properties and print them using document.write method of Javascript

```
<html>
<head>
<script type="text/javascript">
//create a student object
var student ={};
//assign name property
student.name = "Smith";
student.age = "29";
student.hobby = "Coding";
```

```
var val;
for ( val in student) {
 document.write(val);
}
</script>
</head>
</html>
```

functions

Functions are first-class objects in Javascript, means they are objects and can be manipulated and passed around just like any other object.

Defining functions:

```
function name([param1[, param2[, ... paramn]]]) {
   statements
}
```

Example below shows a function which accepts 2 parameter. Inside the body we have used plus(+) operator to concatenate firstName and lastName variable:

```
function displayName(firstName,lastName){
  alert(firstName+lastName);
}
displayName(“Smith”,”Johnson”);
```

Functions with multiple arguments

Javscript lets us define a function where we dont need to write the arguments to be passed. Instead those arguments are made available inside the function body in the form of an predefined array variable named arguments. This lets us create functions capable of taking multile number of arguments.

Below is an example of a function that returns the sum of numbers passed to it. Note that we can pass multiple parameters now.

```
function sum() {

  var i, res = 0;

  var paramsNo = arguments.length;

  for (i = 0; i < paramsNo; i++){

    res += arguments[i];

  }return res;

}    //returns 15

sum(4,5,6);   //returns 33

sum(4,5,7,8,9);
```

Variable Scope in Javascript

JavaScript has two scopes: global and local. A variable declared outside a function definition has a global scope, a variable that is declared inside a function definition has local scope.

Lets see this with an example

```
var name = “smith”;

var i = 1;

if(i == 1){

 var name = “John”;

 alert(“value of name inside block is “+name);

}

alert(“value of name outside block is “+name);
```

Unlike other programming languages like c, c++ the variable scope is not global.Means redeclaring variable name will be actually modifying the global variable name. Hence the variable declared outside the if block will be overwritten and the value of name will change to john.

Variable scope within function

Consider the example below where we redeclare variable name inside the function createScope. In this case a new scope is created and the global variable name is not overridden by local variable.

```
var name = “smith”;

function createScope()

{

   //redeclaring name creates a new scope inside a function.

    var name = “John”;

    alert(“value of name inside block is “+name);

  }


createScope();

alert(“value of name outside block is “+name);
```

Debugging in Javascript

With the recent boom of Javascript, all major browsers come with their own debug tools. Ex: Chorme has chrom thee dev tools. You can access it by shortcut key ctr+shift+i. For firefox you can use firebug extension. We will be discussing about debugging techniques in later sections.

At a basic level, we need to output some statements within the code.For this we use console.log() function. Ex console.log(“debug message”). It displays the debug message in the console window. Note that donot use alert very often because as it blocks the execution of javascript code.

Date Object

Date object is instance of Date() that represents the current moment

We can create Date object is many ways. If no data is provided to the Date() function, it returns the object with current moment date and time.

Ex: var dateOb = new Date();

We can also pass the unix time stamp to get the date object with particular time details.

Ex var dateOb = new Date(14275427715 91);

We can also provide a date string to the Date function.Note that the string should be in RFC 2822 format

Ex var dateOb = new Date('Aug 1, 1999')

Or var dateOb = new Date('Wed, 19 Aug 1999 04:00:00 GMT')

We can also provide a detailed time description as parameters to the Date function.

Ex:

Syntax:

new Date(year, month[, day[, hour[, minutes[, seconds[, milliseconds]]]]]);

Ex: var dateOb = new Date(2015, 4, 1, 1, 10);

Date.UTC()

This method accepts the same parameters as date in RFC 2822 format, and returns the number of milliseconds in Date object since January 1, 1970, 00:00:00, universal time.

The Syntax is

Date.UTC(year, month[, day[, hour[, minute[, second[, millisecond]]]]])

```
var utcDateOb = new Date(Date.UTC(99, 07, 1, 0, 0, 0));
```

Supporting Timezones

Javascript does not have inbuilt support for timezones. When we create a date in Javascript from an ISO string in the site's timezone, the Date object isl automaticallyconverted into the browser's local timezone (which may be different than the site's timezone which is time of server).

Suppose we want to show the time of running a script at the backend to the front end.

In hat case if we send time to javascript from the server and convert it, it will show a local representation of time.

The problem is that the server had its own timezone. Whatever time Date object got, it converted to the local time!..If my server is at New Jersey with time as 7:00 PM, and i opne in webbrowser in India, Javascript date will show the time in the local format as 7:00PM IST. Of course, thats not right.

Ths solution to this problem is to send the UTC time as ISO time string format from the serve.If sent as ISO time string format, Javascript will detect it and convert it to local time format.

See two examples below

```
var utcTimeString = '2015-03-28T13:19:50.712Z';
var nonUtcTimeString = '2015-03-28T13:19:50.712';

var correctLocalTime = new Date(utcTimeString);
var wrongLocalTime = new Date(nonUtcTimeString);
alert(correctLocalTime);
alert(wrongLocalTime);
```

Note that we can use moment-timezone.js library which have good utility functions for time zones

Date object functions

getTime()

Returns the unix time stamp.

getTimezoneOffset()

Returns the time-zone offset in minutes for the current locale.

getDate() Returns day of the month (1-31) for speficied date in local time.

getDay()

Returns day of the week (0-6) for the specified date in local time.

getFullYear()

Returns the year (4 digits..ex 2015) for the specified date in local time.

getHours()

Returns the hours (0-23) for specified date in local time.

getMilliseconds()

Returns the milliseconds (0-999) for specified date in local time.

getMinutes()

Returns the minutes (0-59) for specified date in local time.

getMonth()

Returns the month (0-11) for specified date in local time.

getSeconds()

Returns the seconds (0-59) for specified date in local time.

Date object functions for UTC

getUTCDate()

Returns the day (date) of the month (1-31) for specified date in universal time.

getUTCDay()

Returns the day of the week (0-6) for specified date in universal time.

getUTCFullYear()

Returns the year (4 digits for 4-digit years) for specified date in universal time.

getUTCHours()

Returns the hours (0-23) for specified date in universal time.

getUTCMilliseconds()

Returns the milliseconds (0-999) for specified date in universal time.

getUTCMinutes()

Returns the minutes (0-59) for specified date in universal time.

getUTCMonth()

Returns the month (0-11) for specified date in universal time.

getUTCSeconds()

Returns the seconds (0-59) for specified date in universal time.

Setter functions for date

The following functinos are used to set the value of date.

setDate() setFullYear() setHours() setMilliseconds() setMinutes() setMonth() setSeconds() setTime()

setUTCDate() setUTCFullYear() setUTCHours() setUTCMilliseconds() setUTCMinutes()

setUTCMonth() setUTCSeconds()

Date Conversion functions

toDateString()

Returns the "date" as a human-readable string.

toISOString()

Converts a date to a string following the ISO 8601 Extended Format.

toJSON()

Returns a string representing the Date using toISOString().

toLocaleDateString()

Returns a string with a locality sensitive representation of the date portion of this date based on system settings.

toLocaleFormat()

Converts a date to a string, using a format string.

toLocaleString()

Returns a string with a locality representation of this date.

toLocaleTimeString()

Returns a string with a locality representation of the time portion of this date based on system settings.

toString()

Returns a string representing the specified Date object. Overrides the Object.prototype.toString() method.

toTimeString()

Returns the time of the Date as a human-readable string.

toUTCString()

Converts a date to a string using the UTC timezone .

valueOf()

Returns the primitive value of a Date object.

Math Object

Math is a built-in Javascript object that has properties and methods for mathematical constants and functions.

Note that Math is not a function object, this means Math is not a constructor.

All properties and methods of Math Object are static.We can directly refer to constant pi as Math.PI or call any function as Math.functionName().

Some important constants are Math.PI(Pi constant), Math.Log2E(base 2 e logarithm), Math.Log10E(base10 e logarithm),Math.E(Euler's Constant)

Following are few important functions

sin(), cos(), tan(), asin(), acos(), atan(), atan2(). All these expect values in radians and return values in radians

Math.floor(n): This function returns the largest integer less than or equal to a number..

Math.max([x[, y[, …]]])

Returns the largest of zero or more numbers.

Math.min([x[, y[, …]]])

Returns the smallest of zero or more numbers.

Math.pow(x, y)

Returns base to the exponent power

Math.random()

Returns a random number between 0 and 1.

Math.round(x)

Returns the value of a number rounded to the nearest integer.

String Object

Strings are object in javascript.They are instance of the global String Object which returns a string object.

All string objects have length property that returns the no of characters in a string.

Ex console.log('hello i am sumit'.length);

Note that a single space is also counted.

charAt(index): This function returns the character at the specified index.

Ex 'Smith'.charAt(2) returns **i .** Note that index starts at 0

substr

This method returns the characters in a string beginning at the specified location through the specified number of characters specified in the parameters.

Below is the usage

str.substr(start[, length])

Start denotes the starting of the characters and optional lenght denotes the length to extract.

negative value of start is treated as (string Length + start) value.

Ex:

```
var str = 'Smith Johnson';

console.log(str.substr(1,3)); //returns mit

console.log(str.substr(-4,2));//returns ns
```

String Object vs string primitives

Note that when we can create strings using the String() constructor,It creates an object.

However when we directly refer to a string within quotes, it creates a primitive string type.

When any function or property lookup is called on primitive sting type, JavaScript will automatically wrap the string primitive to call these methods or get properties

TO test run this code in the console

```
console.log(typeof new String('sumit'));

console.log(typeof new String('sumit').valueOf());

 console.log( typeof String('sumit'));

  console.log( typeof 'sumit');.
```

Note that eval should be called on the string primitive to do the code evaluation.

Note that we can use the valueOf function on string object to convert it to primitive string type . This can be useful if we want to use the eval.

Browser Object Model

BOM is Browser Object Model . Its a collection of objects that give us access to

the browser and the computer screen. These objects are attached to the global window object.

window.navigator

The navigator object contains information about the browser and its

Capabilities.

The navigator contains properties like buildId , plug-in list, and few other browser properties.

It also contains important objects. They are navigator.battery and navigator.geolocation. The navigator.battery is a device api that gives information about the battery like battery status(charging or not), battery data etc.

Navigator.geolocation gives information about user's location. We will be discusssing about this more in the HTML5 slides.The navigator object contains information about the browser and its

capabilities.

window.history

It allows access to the previously visited pages for a specific browser session.

For example, We can see how many pages the user has visited

before coming to your page using window.history.length

Window.history.back() and window.history.forward() can take the user to back and forward urls.

Similary we can also use window.history.go(-2) to go to 2 pages back.

Window.history.go(0) will reload the same page.

window.location

This property points to an object that contains information about the URL

of the currently loaded page.

Location.href represents the page url and location.hostname represents the hostname.

To find all the properties we can iterate through the location array and print its value in the console.

location.hash represents the hash url data. Ie www.mysite.com/test.php#top. the value of hash property here will be top.

Location.port gives the port value.

```
for (var index in location) {

if (typeof location[index] === "string") {

console.log(i + ' = "' + location[i] + '"');

}

}
```

Note that we can change the webpage using the following ways.

location.href = "http://yahoo.com";

location = "http://yahoo.com";

location.assign("http://yahoo.com");

location.replace("http://sify.com");

Note that using location.replace we can navigate to a url without creating entry in browser history. Location.reload is used to reload the page.ie location.reload();

What happens if we remove the http:// and directly run like location.href = “yahoo.com”;

window.frames

It represnets a collection of all of the frames in the current page.

frames.length returns the no of frames in the page.

Note that each frame contains another page, which has its own global window object which has its own set of properties.

String Object vs string primitives

Note that when we can create strings using the String() constructor , It creates an object.

However when we directly refer to a string within quotes, it creates a primitive string type.

When any function or property lookup is called on primitive sting type, JavaScript will automatically wrap the string primitive to call these methods or get properties

TO test run this code in the console

```
console.log(typeof new String('sumit'));

console.log(typeof new String('sumit').valueOf());

 console.log( typeof String('sumit'));

  console.log( typeof 'sumit');.
```

Note that eval should be called on the string primitive to do the code evaluation.

Note that we can use the valueOf function on string object to convert it to primitive string type . This can be useful if we want to use the eval.

Working with DOM

The Document Object Model (DOM) represents representation of XML /HTML document as a tree of nodes. Using DOM we can access any html element on the page, modify, remove or add new elements. Note that DOM is an API (Application Programming Interface) and can be implemented other languages also.

When HTML code loads in the browser, the entire HTML document is parsed and every html node is represented as an object. Every node object has properties and functions to read and change the properties . In the example below, have a look at the paragraph tag.

Have a look at code below. Note that <h3> tag, tag are dom objects with parent <p>. the tag . The div and h3 tag are siblings as they are at same level. Using Dom we can now access and manipulate the properties of any of the tags below . Note that <html> is the root of the document tree.

```
<html>
        <head>
        <title>Welcome to AcadGild</title>
    </head>
    <body>
      <p>
            <h3> welcome to AcadGild</h1>
        <div> Lets move ahead    </div>
      </p>
    </body>
  </html>
```

DOM Manipulation

DOM API exposes many functions to manipulate with HTML. Lets have a look at important functions.

First to access any HTML element, we need to get access to the corresponding DOM object reference. We can use document.getElementByID to get a reference.

This function expects the id attribute as parameter. Consider the example below. Once we get the reference, we have access to all the properties of the node. One such property is textContent which returns the text inside the node. Note that if the node has children, then it returns the contant of the node and content of children inside.Note that we have other property namely innerText that returns the text. But innerText property is very slow and hence stick to textContent property to get the text.

```
<html>

<body>

<div id = “outer”> Hello </div>

<script type = “text/javascript”>

var ref = document.getElementById(“outer”);

alert(“the content inside div tag is “+ref.textContent);

</script>

</body>

</html>
```

DOM Manipulation Functions

Following are the other functions to get an element reference

document.getElementsByTagName(tagname) :This method returns a collection of all elements reference in the document with the specified tag name.

document.getElementsByClassName(classname):This method returns a collection of all elements reference in the document with the specified class name.

Note that the return value of all the above functions has a length property which returns the length of the elements references returned . Have a look at the following example. The following example displays the content inside and <p> tag using the getElementsByClassName function.

```
<head><script type = “text/javascript”>

function init(){

  var refs = document.getElementsByClassName("container");

  var count = refs.length;

  var i = 0;

 for(i=0;i<count;i++) {

    alert(refs[i].textContent);

  }

}
```

```
</script></head><body onload =”init()” >
<div class="container">welcome to the
site</div>
<p class="container"> Welcome to the page</p>
</body>
```

Modifying HTML using innerHTML

InnerHTML is the property of DOM object nodes. Using this property we can get/set the html inside a tag. Lets change the html inside a tag using this property in the example below.Note that we have added a onclick attribute which adds an event listener of click on the button. The value of onclick attribute is the function to execute on click of button. We will discuss about events in the coming section.

```
<head>
<script type = "text/javascript">
  function addHeading(){
    var ref = document.getElementById("container");
    var htmlToInsert= '<h3> This is the Heading</h3>';
    ref.innerHTML = htmlToInsert;
  }
</script></head><body>
<button onclick = "addHeading()">Add Heading</button>
<div id="container"></div>
</body>
```

Events

Javascript can also respond to events which can also be actions by the user. For example clicking on a element, hovering over an element are all actions by user and JavaScript uses events can react to these actions.

Javascript attaches a function called an event listener or event handler to a specific event and the function invokes when the event occurs.

Events can be attached by the following ways

1)Inline HTML attributes

2)Adding to element properties with javascript

3)Using DOM Event Listeners

Inline event listeners

Events can be attached as attributes to the elementsn like this

Click

In the above code, if any area on the div is clicked, showMsg function is fired. Though this method is the easiest, we should not use this method because it makes the html code messy.

Adding Events Elements properties

We can also assign a function to the onclick property of a DOM node element.Have a look at the code snippet below

```
<div id = “container”>click here</div>

<script type = “text/javascript”>var ref = document.getElementById('container');

ref.onclick = function () {

alert('The div area is clicked');

};

</script>
```

Note that this is more cleaner than the inline attribute approach discussed previously

Using DOM Event Listeners

The best way to handle events is to use the event listener approach.

We can assign listeners to the click event using theaddEventListener() method.Below is the format

ref.addEventListener(event,function)

In the example below we have added 2 functions to be executed on click on a div element

```
<div id = "container">click here</div>
<script type = "text/javascript">
  var ref = document.getElementById('container');
  ref.addEventListener('click',showMsgOne);
  ref.addEventListener('click',showMsgTwo);
  function showMsgOne(){
    alert("THIs is message one");
  }
  function showMsgTwo(){
    alert("THIs is message Two");
  }
</script>
```

List of events

Mouse Events:

mouseup, mousedown, click dblclick,

mouseover (mouse over an element),

mouseout (mouse was over an element and left it),

mousemove

Keyboard events

keydown, keypress, keyup

window events

load (an image/ page and all of its components are done loading),

unload (user leaves the page),Beforeunload ,error (JavaScript error) ,

abort (user stops loading the page),

resize (browser window resize event),

scroll (the page is scrolled),

contextmenu (appearance event of right-click menu)

Form events

focus (on enter of form field),

blur (on leave of form field)

change (on leave of a field after the value has changed),

select (select text in a text field)

reset (wipe out all user input),

submit (send the form)

Ajax

AJAX means Asynchronous JavaScript and XML. it is the communication to the server side making use of the XMLHttpRequest object. we can get text,json,xml or plain strings from the server using ajax

To make ajax request we need to get the httprequest object.

```
var httpRequest;
if (window.XMLHttpRequest) { // For Mozilla, Safari, IE7+
    httpRequest = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE 6 and older version
    httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
}
httpRequest.onreadystatechange = runTheFunction;
```

This functin runs when there is a change of state

Below is the list of the readyState values

- **0 (uninitialized request)**
- **1 (loading)**
- **2 (loaded)**
- **3 (interactive)**
- **4 (complete)**

Concluding

We have explored the syntax of javascript and how to execute it on the web page.. Javascript can do many operations like DOM modification, Form Validation,Events and others. However It is not recommeded to do all these common operations with core Javascript because of complicated code and cross browser issues. In the next section we will be learning Jquery, which is the recommended library for common scripting tasks on the web page