JavaScript - Basics

- JavaScript Overview
- JavaScript Building Blocks
- JavaScript Objects

JavaScript Overview

Client-side programming

- Client side Programming is done using HTML and some scripting languages like JavaScript or VBScript.

- Recall: HTML is good for developing *static* pages
  - can specify text/image layout, presentation, links, …
  - Web page looks the same each time it is accessed

- In order to develop interactive/reactive pages, must integrate programming or scripting language.

What is JavaScript? (1/2)

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- Unlike HTML, It is case sensitive.
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

What is JavaScript? (2/2)

- **Are Java and JavaScript the Same?**

NO!

JavaScript is to java as Carpet is to Car.

JavaScript was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all Netscape and Microsoft browsers since 1996.

What can JavaScript do?

- JavaScript gives HTML website designers a programming tool.

- JavaScript enables you to
  - create dynamically updated content,
  - control multimedia,
  - animate images,
  - validate Data,
  - detect browser
  - Perform event handling
  - And much more…..

Hello World Example - JavaScript in HTML

```
<html>
<body>
<script type="text/JavaScript">
document.write("Hello World!");
</script>
</body>
</html>
```

The code above will produce this output on an HTML page:

```
Hello World!
```

```
<html>
<body>
<script type="text/JavaScript">
document.write("<h1>This is a header</h1>");
</script>

</body>
</html>
```

The code above demonstrates that HTML tags can be used within JavaScript code

Add JavaScript to HTML Pages

- **Internal JavaScript**
  - <script> tag in the HTML page

```
<script type="text/JavaScript">
  function message()
  {
   alert("This alert box was called with the onload event");
  }
</script>
```

- **External JavaScript**
  - Create a file with a '.js' extension.
  - Add the JavaScript code in this file without the <script> tag
  - Add a script tag in the HTML file as shown below:

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

- **Inline JavaScript Handlers**
  - Used along with the HTML tag only
  - For example:

```
<button onclick="alert('Button Clicked')">Click Me</button>
```

Troubleshooting JavaScript

- Types of Errors
  - Syntax Errors
  - Logical Errors

- Use Developer Tools JavaScript console to identify and resolve syntax errors.

JavaScript Building Blocks

- DataTypes and Variables
- Operators
- Conditional Statements
- Loops
- String in JavaScript
- Arrays
- Functions
- Event Handling

Data Types and Variables

- Data Types in JavaScript
    - Numbers
    - Strings
    - Boolean
    - Arrays
    - Objects

Declaring a Variable

- Creating variables in JavaScript is most often referred to as "declaring" variables.
- You can declare JavaScript variables with the **var** statement:
  - var x;
  - var car_name;

- JavaScript is a loosely typed language, we don’t need to specify what data type a variable will contain.
  - var myString = ‘Hello’;
    - will automatically assign String data type to the variable myString

- If you assign values to variables that have not yet been declared, the variables will automatically be declared. These statements:
  - x=5;
  - car_name="Volvo";

have the same effect as:

  - var x=5;
  - var car_name="Volvo";

Declaring Variables

- Rules for variable names:
  - Variable names are case sensitive
  - They must begin with a letter or the underscore character
  - Keywords and reserved words cannot be used for naming

- Lifetime of Variables
  - When you declare a variable within a function, the variable can only be accessed within that function.
  - When you exit the function, the variable is destroyed.
  - These variables are called local variables.

  - If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

JavaScript Operators

- Arithmetic

  +, -, *, /, %

- Increment and Decrement operators

  ++, --

- String concatenation operator

  +

- Assignment

  +=, -=, *=, /=. %=, =

- Comparison

  === (strict equality), !==(strict non-equality) , ==, !=, >,<,>=,<=

- Logical Operators

  &&, ||, !

More on JavaScript Operators

- As in C++/Java, precedence rules apply to expressions

  (* / %)→(+ -)→(&& || !)

- Operators are left-associative (evaluated in left-to-right order)

- Be careful when mixing strings and numbers. The rules are:
  - number + number ➔ addition
  - string + string ➔ concatenation
  - string + number ➔

    convert number to string, then concatenation

Mixing up Data Types

```
S=“abc”;
I=123;
alert(S+I);         → abc123

B=true;
alert(S+B);         → abctrue

alert(B+I);           → 124

alert(5+154e-2 +” kgs”);     →6.54 kgs

alert(“Rs.” +5+154e-2 );     →Rs .51.54

alert("34"==34);  ⎫
                  ⎬ → true
alert(1<".23");   ⎭

alert(10*"55");     → 550
```

Few more

```
alert(10/"abc");        → NaN

alert(1==true);         → true

alert(1=="true");       → false

alert(5/0);             → Infinity

alert(true & "3");      → 1

alert(true & "a");      → 0
```

Conditional Statements

- if statement
  - use this statement if you want to execute some code only if a specified condition is true
- if...else statement
  - use this statement if you want to execute some code if the condition is true and another code if the condition is false
- if...else if....else statement
  - use this statement if you want to select one of many blocks of code to be executed
- switch statement
  - use this statement if you want to select one of many blocks of code to be executed

Loops

- for
  - loops through a block of code a specified number of times
- While
  - loops through a block of code while a specified condition is true
- Do-while
  - loops through a block of code atleast once even if the specified condition is false

- Creating string

  var str = “Welcome”;

  str = “Welcome”;

  str = ‘Welcome’;

Functionality	Function	Example
Escaping Character	\’	

String concatenation	+	var one = 'Hello, ';
		var two = 'how are you?';
		var joined = one + two;

Converting a string containing digits to number	Number(str)	Var str = ‘123’; Var myNum = Number(str);
Converting number to string	myNum.toString()	Var myNum = 123; Var str = myNum.toString()

Arrays

- Declaration: arrays are objects in JavaScript

```
arr = new Array(3);
```

- A single array can hold any kind of data

```
junk= new Array(3);
junk[0]="Sunday";
junk[1]=0       ;
junk[2]=true;
```

- Initialized array

```
Week = new Array("sun","mon","tue","wed","thu","fri","sat");
alert(week[0]); →sun
alert(week); →sun,mon,tue,wed,thu,fri,sat
```

- Array length:

```
a.length→ 3
```

Adding elements to an array

- Add using subscript. Array size is incremented dynamically

```
    A = new Array();
    a[4]=4;
    a.length is 5
```

- **`push()`** to automatically adds elements to the end of the array

```
week=new Array(“sun”,”mon”,”tue”,”wed);
week.push('thu');
week.push('fri','sat');
alert(week); →sun,mon,tue,wed,thu,fri,sat
```

- **`unshift`** to automatically adds elements to the beginning of the array

```
nums=new Array(3,4,5,6);
nums.unshift(0,1,2);
alert(nums);   -> 0,1,2,3,4,5,6
```

Removing elements from an array

- **pop()** to remove element from the end of the array

```
week=new Array("sun","mon","tue","wed");
week.pop();
alert(week);→sun,mon,tue
```

- **shift()** to remove element from the beginning of the array

```
nums=new Array(3,4,5,6);
nums.shift()
alert(nums);→ 4,5,6
```

Converting between strings and arrays

- split converts a string into array

```
week = “sun,mon,tue,wed,thur,fri,sat”;
var myArray = week.split(‘,’);
```

- Join converts an array into string

```
var newString = myArray.join();
```

Iterating over an Array

- For loop
- For .. In loop

Example:

```
var arr = [“red”, “green”, “blue”];
arr[5] = “yellow”;

for( x = 0, len = arr.length; x < len; x ++){
        document.write(arr[i] +”,”);
}
```

Output:

```
red, green, blue, undefined, undefined, yellow
```

```
for( x in arr){
        document.write(x + “,”);
}
```

Output:

```
red, green, blue, yellow
```

Functions

- Functions perform specific tasks and can be used repetitively throughout the program.
- Some features of the functions are:
  - Reusability
  - Modularity
  - Overall programming simplicity
- Built-in functions
  - Built-in functions are already coded to ease the programmer's task.
  - Some built-in functions supported by JavaScript are:
    - isNaN(): Helps determine if an argument is NaN (not a number).
    - parseInt(): Parses the string argument and returns an integer.
    - parseFloat(): Accepts a string argument and returns a floating point number.
    - typeOf(variable): gives the data type of the variable defined here/
- User-defined functions
  - Function defined in code to enhance readability and efficiency

User-Defined functions

- Syntax:

```
function function_name(arg1, arg2,..,argx){
//function code
//return statement (optional)
}
```

- Example:

```
function display(name){
        alert("Hello" + name);
}
```

Calling a function

- The above function can be called as
  - **`display();`**
  - **`display(“hello”);`**
  - Or **`display`** with any number of arguments

```
function display(x){
  if(x==null)
    x=“Greetings”;
    alert(x) ;
}
```

- You can also pass values to a function that does not take any arguments!

Anonymous Functions

- Functions without a name

```
function(){
        alert(‘Hello’);
}
```

- Anonymous functions are generally used with event handlers, for example:

```
myButton.onClick = function(){
        alert(‘Hello’);
}
```

- Anonymous function can be assigned to be the value of a variable, for example:

```
var myGreeting = function() {
        alert('hello');
}

//Calling of the function
        myGreeting();
```

Events

- In JavaScript, events are actions that occur as a result of user interaction with the Web pages or other browser-related activities.

- Few Examples of events are:
  - The user clicking the mouse over a certain element, or hovering the cursor over a certain element.
  - The user pressing a key on the keyboard.
  - The user resizing or closing the browser window.
  - A web page finishing loading.
  - A form being submitted.
  - A video being played, or paused, or finishing play.
  - An error occuring.

- Each available event has an event handler, which is a block of code that will run when the event fires.

- This block of code has to be defined to be run in response to an event firing, we say we are registering an event handler.

Few Event Examples

Attribute	The event occurs when...
Onabort	Loading of an image is interrupted
onblur	An element loses focus
onchange	The user changes the content of a field
onclick	Mouse clicks an object
onfocus	An element gets focus
onkeydown	A keyboard key is pressed
onkeypress	A keyboard key is pressed or held down
onkeyup	A keyboard key is released
onload	A page or an image is finished loading

Few Event Examples

Attribute	The event occurs when...
onmousedown	A mouse button is pressed
onmousemove	The mouse is moved
onmouseout	The mouse is moved off an element
onmouseover	The mouse is moved over an element
onmouseup	A mouse button is released
onreset	The reset button is clicked
onselect	Text is selected
onsubmit	The submit button is clicked
onunload	The user exits the page

Ways of using Web Events

- Event Handler properties
- Inline Event Handlers
- Registering Event listeners (addEventListener(), removeEventListener())

Event Handler properties

- Every element in an HTML document have an event handler property which can be used to define the event handler.
- For example:

```
var btn = document.getElementById(“btn1”);
btn.onclick = function(){  //code to be performed }
```

or we can also set the handler property to be equal to a named function name

```
function change(){
        //some code
}

btn.onclick = change;
```

Inline event handlers

- Used along with the HTML element definition

```
<button onclick="bgChange()">Press me</button>
```

- This is the earliest and easiest method of registering event handlers
- But using them is considered bad practice because of the below reasons
  - Its not good idea to mix HTML and JavaScript, as it becomes hard to parse
  - It becomes unmanageable and inefficient if lot of event handling is required in almost all of the components.

Registering Event listeners

- Newest type of event mechanism defined in DOM Level 2 events

- Example:

```
var btn = document.getElementById(“btn1”);

function change(){  //some code }

btn.addEventListener(‘click’, change);
```

- To remove an event listener

```
btn.removeEventListener(‘click’, change);
```

- Using this way, we can add multiple handlers to the same event

```
btn.addEventListener(‘click’, functionA);
btn.addEventListener(‘click’, functionB);
```

-> Both the functions will run when the element is clicked.

Which mechanism should be used?

- Event handler properties have less power and options, but better cross browser compatibility (being supported as far back as Internet Explorer 8).
- DOM Level 2 Events (addEventListener(), etc.) are more powerful, but can also become more complex and are less well supported (supported as far back as Internet Explorer 9).
  - The main advantages of the this mechanism are that you can remove event handler code if needed, using removeEventListener(), and you can add multiple listeners of the same type to elements if required.

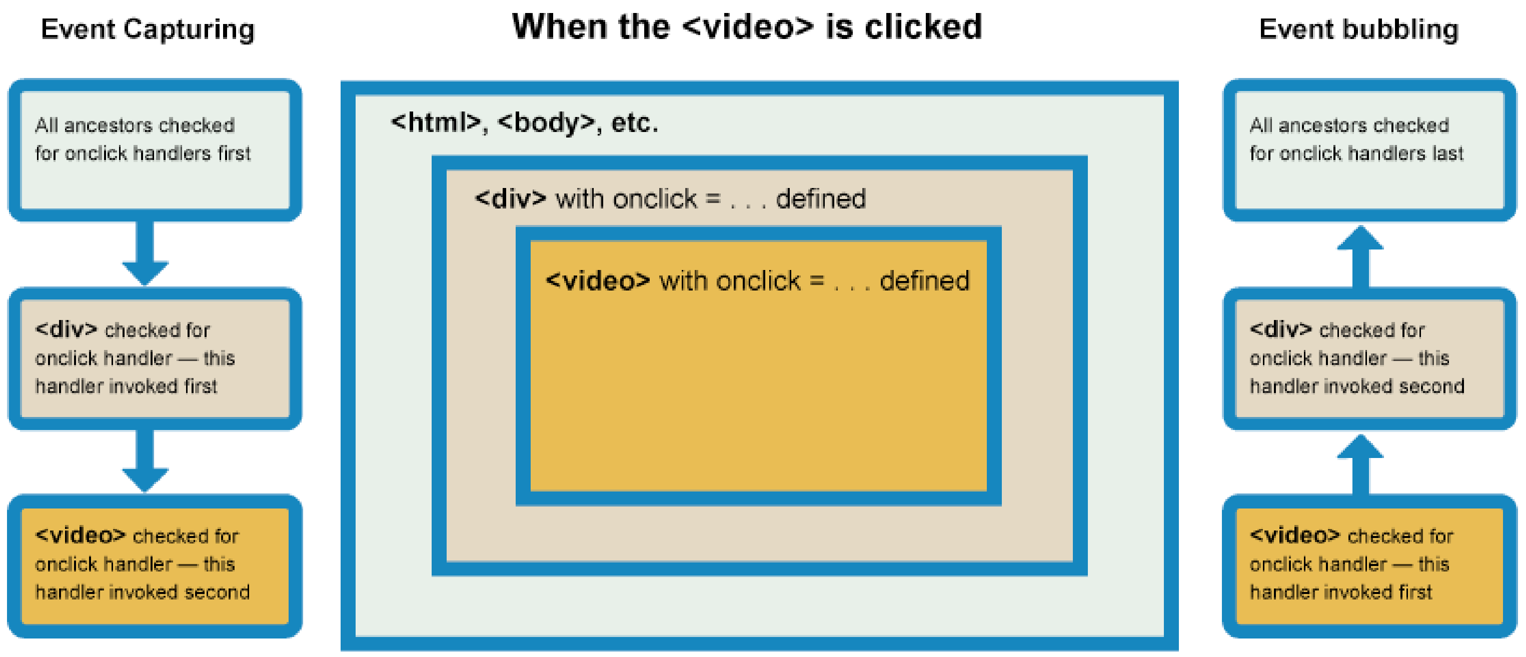
Event Object

- Event object is created as soon as an event is generated and it is automatically passed to the event handler.
- Event object contains information about the event which has been generated.
- Some of the properties/Methods of Event object are:
  - type
  - target
  - preventDefault()
  - stopPropagation()
- For every event type, there are specific properties and methods attached to them. For example, MouseEvent also have properties like pageX, pageY, shiftKey, altKey etc.
- For more details, refer to : https://developer.mozilla.org/en/docs/Web/API/Event

Event Bubbling and capture

- When an event is fired on an element that has parent elements, two phases are executed:
  - The capturing phase
  - The bubbling phase
- In the capturing phase:
  - The browser checks to see if the element’s outer-most ancestor(<html>) has an onclick event handler registered on it in the capturing phase and runs it if so
  - Then it moves on to the next element inside <html> and does the same thing, then the next one, and so on until it reaches the element that was actually clicked on
- In the bubbling phase:
  - The browser checks to see if the element that was actually clicked on has an onclick event handler registered on it in the bubbling phase, and runs it if so.
  - Then it moves on to the next immediate ancestor element and does the same thing, then the next one, and so on until it reaches the <html> element.

Event Bubbling and capturing


Event Capturing
When the <video> is clicked
Event bubbling
All ancestors checked for onclick handlers first
<div> checked for onclick handler — this handler invoked first
<video> checked for onclick handler — this handler invoked second
<html>, <body>, etc.
<div> with onclick = . . . defined
<video> with onclick = . . . defined
All ancestors checked for onclick handlers last
<div> checked for onclick handler — this handler invoked second
<video> checked for onclick handler — this handler invoked first

JavaScript Objects

JavaScript: Object Oriented programming

- JavaScript is an Object Oriented Programming (OOP) language. An OOP language allows you to define your own objects and make your own variable types.
- OOPs Principles
  - Abstraction
    - Representing only the most important aspects for the program purpose
  - Encapsulation
    - Data and methods are stored together in an object package making it easy to structure and access
  - Polymorphism
    - One name, multiple functionalities
    - Not supported in JavaScript
  - Inheritance
    - Object inherits the properties and methods from the parent and add specialized features directly on them as needed.

Object Basics

- An Object is an entity having some properties (data) and behavior(functionality) associated with it.
- **Properties (data)**
  - Properties are the values associated with an object.
  - In the following example we are using the length property of the String object to return the number of characters in a string:

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

- **Methods (functionality)**
  - Methods are the actions that can be performed on objects.
  - In the following example we are using the toUpperCase() method of the String object to display a text in uppercase letters:

```
var str="Hello world!";
document.write(str.toUpperCase());
```

Defining an Object template

- Object template (also called as Class) defines what characteristics an object should have.
- We can create object instances from this object template.
- Multiple objects created using the same object template have the same properties (values can be different) and same methods.
- JavaScript uses special functions called **constructor functions** to define objects and their features.
- Constructors provide the means to create as many objects as needed in an effective way, attaching data and functions to them as required.
- Example:

```
function Person(name){
        this.name = name;
        this.greeting = function(){
                alert('Hi, I\'m ' + this.name + '.');
        }
}
```

Creating Object instances

- Objects are created using the below syntax:

  var person1 = new Person(“Amit”);

  var person2 = new Person(“Raj”);

  -> we are creating two objects

  -> we use ‘new’ keyword to create the new object instance.

- The ‘this’ keyword
  - The ‘this’ keyword refers to the current object.
  - ‘this’ helps objects in using their own values, and not some other value

Accessing Object properties and methods

- Dot Notation
  - To access properties or methods, Dot (.) notation is used.
  - Example:
    - to refer to property - **person1.name**
    - to refer to method - **person1.greeting()**

- Bracket Notation
  - To access properties, we can use Bracket notation as well.
  - Example:

    person1[‘name’]

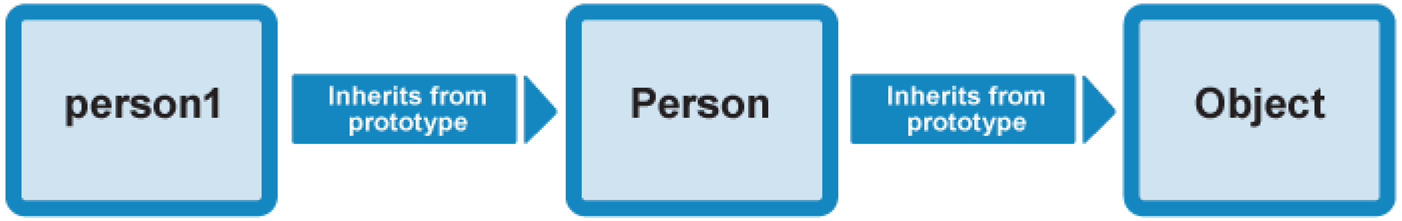
- We can also create new members in the object space using these notations.

```
person1[‘age’] = 40;
person1.farewell = function(){ alert(‘Bye’); }
```

-> After execution of these two statements, property ‘age’ and method ‘farewell()’ is added to person1 object

Object Prototypes

- Prototypes are the mechanism by which JavaScript objects inherit features from one another
- Each object in JavaScript has a prototype object, which acts as template object that it inherits methods and properties from.
- An object’s prototype object may also have a prototype object, which it inherits methods and properties from, and so on. This is referred as **prototype chain** as shown below:



Working with JSON

- JSON is a standard format for representing structured data as JavaScript objects
- JSON can be used for representing and transmitting data on web sites
- Example of a JSON object:

```
var person = {
        “name” : “Neelam”,
        “age” : 30,
        “hobbies” : [ “Music”, “Cricket”, ”Collecting Stamps” ],
        “company” : {
                “name”: “Aricent”,
                “location”: ”Gurgaon”,
                “dateOfJoining” : “12-Sep-2015”
        }
}
```

- To access the JSON object values

```
person.name
person.company.name
```

Important Points related to JSON

- JSON can exist as an object, or a string. Object is used to read data out of the JSON and string is used to send the JSON across the network
- JavaScript provides a global JSON object containing methods for converting between the two
- Methods are:

  -> JSON.parse()

  - Parse a string as JSON, optionally transform the produced value and its properties, and return the value

  -> JSON.stringify()

  - Return a JSON string corresponding to the specified value, optionally including only certain properties or replacing property values in a user-defined manner.

Important points related to JSON

- JSON is purely a data format — it contains only properties, no methods.
- JSON requires double quotes to be used to be valid.
- Even a single misplaced comma or colon can cause a JSON file to go wrong, and not work. You can validate JSON using an application like JSONLint.
- JSON can actually take the form of any data type that is valid for inclusion inside a standard JSON object, not just arrays or objects. So for example, a single string or number would be a valid JSON object.
- Unlike in JavaScript code in which identifiers may be used as properties, in JSON, only strings may be used as properties.