

MEAN Stack



Harshita

Introduction

- MEAN is an opinionated full stack JavaScript framework which simplifies and accelerates web application development.
- MEAN represents a major shift in architecture and mental models – from relational databases to NoSQL and from server-side Model-View-Controller to client-side, single-page applications.
- MEAN is a acronym for MongoDB, ExpressJS, Angular and Node.Js.

Problems with XAMPP?

- Apache is not the fastest web server around
- It's hard to write good-to-read, reusable and fast PHP code
- Frontend works with other languages than the backend
- Too many conversions (XML to PHP to HTML, model to SQL)
- There is no separated server-side and client-side development

Requirements for a modern web?

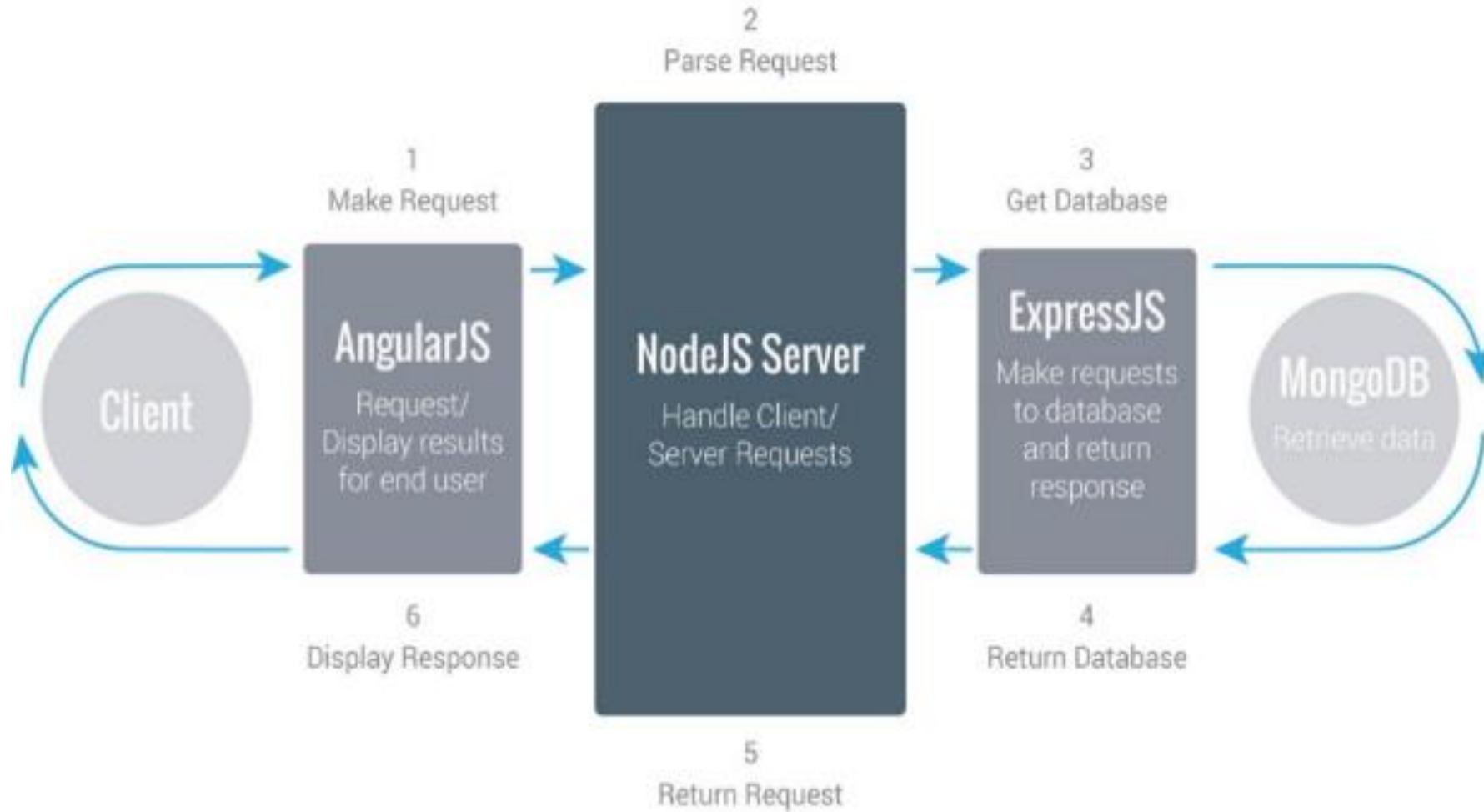
- Customer want fast websites/fast response times
- No page reloads
- Enterprises want to go virtual
 - one box + Several virtual images=>Shared Hardware
 - System with minimal memory footprint/overhead needed
- As many concurrent requests as possible
- Only load resources when needed (conditional loading)
- Mobile/ Responsive UIs

What is MEAN Stack?

- MEAN Stack is a full-stack JavaScript solution that helps you build fast, robust and maintainable production web applications using MongoDB, Express, Angular and Node.js.
- 100% free, 100% Open Source
- 100% JavaScript (+JSON and HTML)
- 100% Web standards
- Consistent Models from the backend to the frontend and back

- Use a uniform language throughout your stack
 - JavaScript (the language of the web)
 - JSON (the data format of the web)
 - No conversion needed for the database
- Use JavaScript with a great framework (compared to jQuery)
- Allow to start with the complete frontend development first
- Very low memory footprint/overhead

Processing Model



MongoDB

- Fast NoSQL schemaless database written in C++

MongoDB	RDBMS
Collection	Table
Document	Row
Index	Index
Embedded Document	Join
Reference	Foreign Key

Benefits of MEAN Stack Development

- It is Flexible
- Allow Isomorphic Coding
- It's JavaScript Everywhere
- Excellent performance
- Increased developer flexibility and efficiency
- Reduced development cost
- Easier debugging and code-reuse

Session-1

Topics to be covered...

- What is Scripting Language
- Client vs. Server Side Scripting
- Introduction to JavaScript
- Where JavaScript is used
- Variables in Javascript
- JavaScript Console
- Let keyword
- Use Strict Keyword
- Javascript Hoisting
- JS Operators
- Function
- Control Structure and loops
- Write(),alert(),confirm() and prompt() box
- DOM Events
- Global Properties and methods

A Script is a program or sequence of instructions that is interpreted or carried out by another program rather than by computer processor.

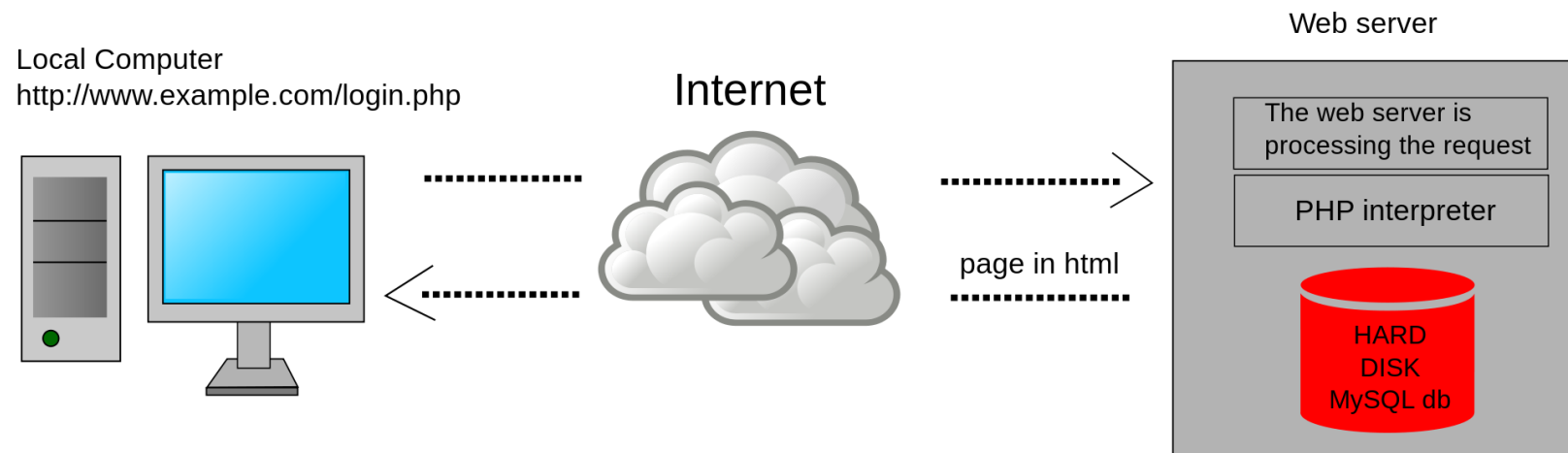
The scripting language is basically a language where instructions are written for a run time environment. They do not require the compilation step and are rather interpreted.

Server Side Scripting

Server side scripting is used to connect to the database that reside on the web server.

Server side scripting can access the file system residing at the web server.

Response from a server-side script is slower as compared to a client-side script because the scripts are processed on the remote computer.



Advantage: Your scripts are hidden from view. Users only see the HTML output , even when they view the source.

Client Side Scripting

Program that execute on client side, by the web browser instead of server side.

Upon request, the necessary files are sent to the user's computer by the web server on which they reside.



Advantages:

Allow for more interactive by immediately responding to user actions.

Execute quickly because they don't require a trip to the server.

Can give developers more control over the look and behavior of their web apps.


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

Tells where the JavaScript starts

Commands for writing output to a page

Tells where the JavaScript ends

This code produce the output on an HTML page:
Hello World!

What is ECMAScript?

ECMAScript (European Computer Manufacturers Association) is a scripting language standard and specification

JavaScript

Jscript

ActionScript

What is JavaScript?

JavaScript is used to program the behaviour of web pages (performing dynamic tasks).

JavaScript are scripts (code) that is executed on the client's browser instead of the web-server (Client-side scripts).

JavaScript is lightweight and cross-platform and loosely coupled.

JavaScript was created by Brendan Eich, a Netscape Communications Corporation programmer, in September 1995.

- **HTML**

- Hypertext Markup Language
- Structure of Page



- **JavaScript**

- Interactivity with User
- Dynamic Updates in a Web Page



- **CSS**

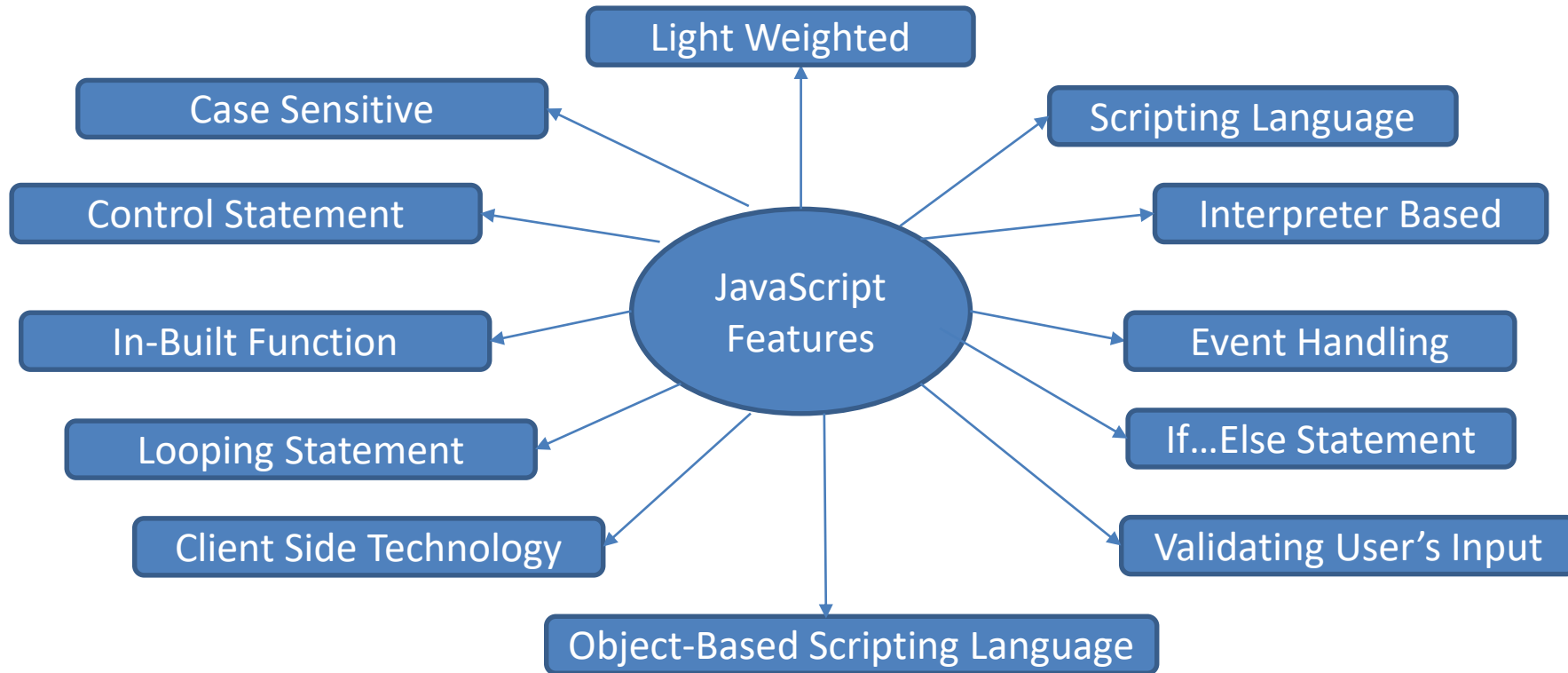
- Cascading Style Sheets
- Presentation/Styling



Why we need client side programming

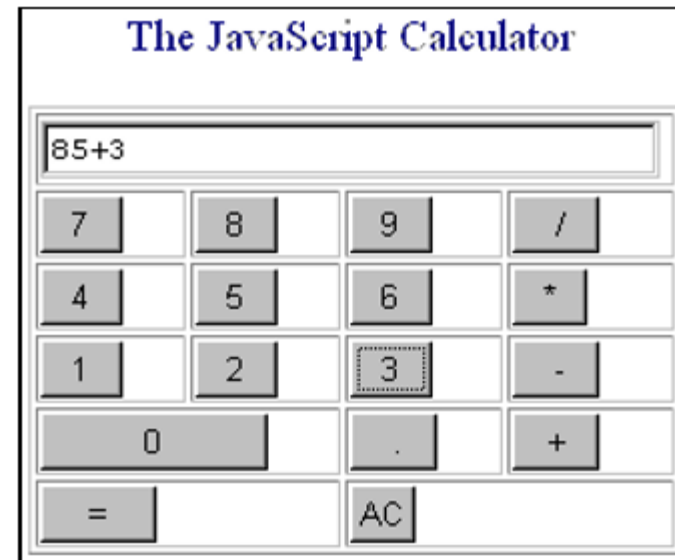
- The user's actions will result in an immediate response because they don't require a trip to the server.
- Allows the creation of faster and more responsive web applications.
- Make web pages more interactive
- Fewer resources are used and needed on the web-server.

JavaScript Features



What Javascript can do?

- Javascript can change HTML Content
- Javascript can change HTML Attributes
- Javascript can change HTML Styles (CSS)
- Javascript can validate Data
- Javascript can Make Calculations



Embedding JavaScript in HTML

1. Anywhere in the html file between <script></script> tags.

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

<h2>JavaScript Demo</h2>

<p id="demo"></p>

<script>
document.write("My First JavaScript");
</script>
|
</body>
</html>
```

2. In an external file and refer to it using the SRC attribute.

```
<!DOCTYPE html>
<html>
<head>
<title>External Javascript</title>
<script src="myScript.js"></script>
</head>
<body>
<p id="demo">A Paragraph.</p>
</body>
</html>
```


JavaScript Display Possibilities

JavaScript can "display" data in different ways:

1. Writing into the browser console, using **console.log()**.
2. Writing into the HTML output using **document.write()**.
3. Writing into an alert box, using **window.alert()**.
4. Writing into an HTML element, using **innerHTML**.

Console Object

The Console object provides access to the browser's debugging console.

Console Object Methods:

log() method: writes a message to the console.

Syntax

`console.log(message)`

```
<!DOCTYPE html>
<html>
<body>
<script>
console.log("Hello world!");
</script>
</body>
</html>
```

table() Method: writes a table in the console view.

```
<!DOCTYPE html>
<html>
<body>
<script>
console.table(["Audi", "Volvo", "Ford"]);
</script>
</body>
</html>
```

VM1440:4

(index)	Value
0	"Audi"
1	"Volvo"
2	"Ford"

► Array(3)

```
<!DOCTYPE html>
<html>
<body>
<script>
var car1 = { name : "Audi", model : "A4" }
var car2 = { name : "Volvo", model : "XC90" }
var car3 = { name : "Ford", model : "Fusion" }

console.table([car1, car2, car3]);
</script>
</body>
</html>
```

VM116:1:6

(index)	name	model
0	"Audi"	"A4"
1	"Volvo"	"XC90"
2	"Ford"	"Fusion"

► Array(3)

clear() Method: clears the console.

The `console.clear()` method will also write a message in the console: "Console was cleared".

Syntax

```
console.clear()
```

Alerts

An alert box is often used if you want to make sure information comes through to the user.

Syntax

```
window.alert("sometext");
```

```
alert("I am an alert box!");
```

An embedded page on this page says

I am an alert box!

OK

Prompts and Confirm

Prompts :The return is the data the user entered

```
<script type="text/javascript">  
window.prompt('Message', 'Initial Value');  
</script>
```

Confirm: The confirm returns true and false

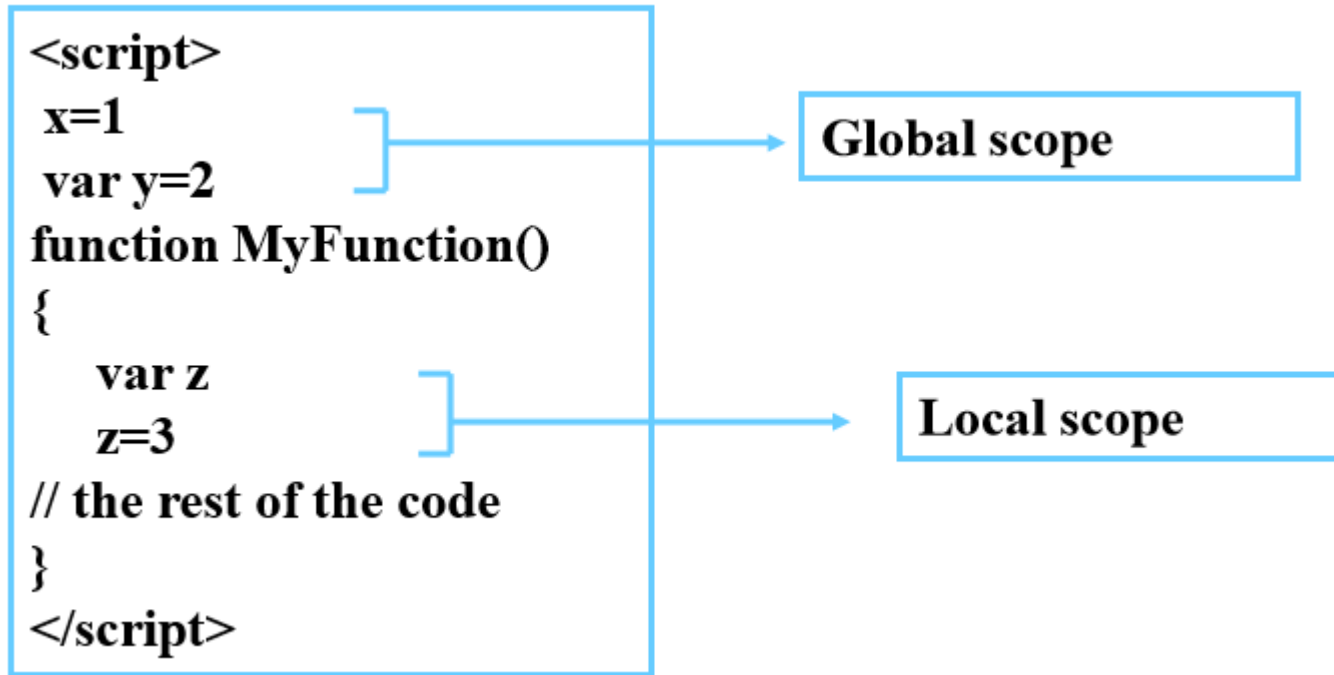
```
<script type="text/javascript">  
window.confirm('Message');  
</script>
```

Variables

JavaScript variables are containers for storing data values..

Variables are declared with the '**var**' keyword.

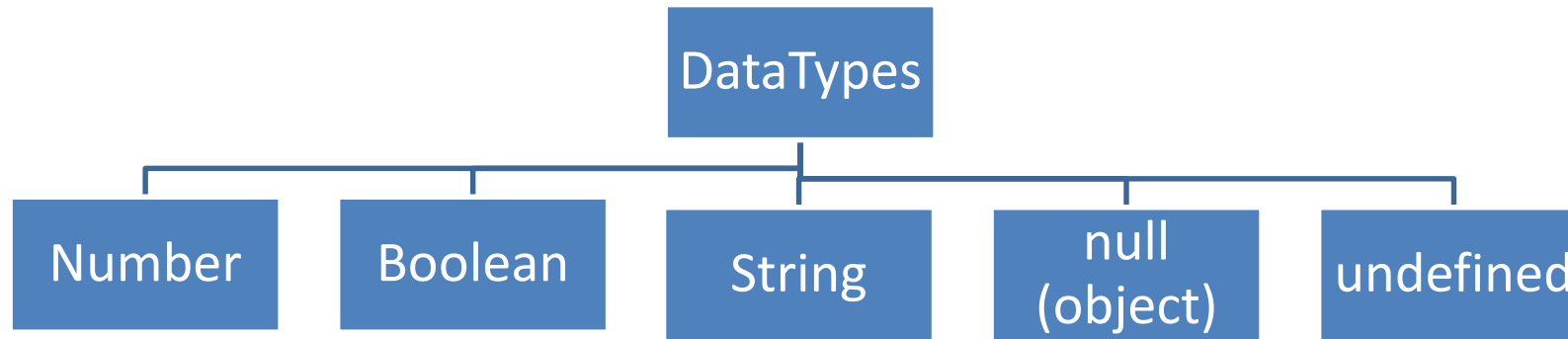
```
var a=10;
```



Lexical Scoping:

Depending on position meaning changes.

Data Types



```
var x=10;
alert(typeof(x));
```


undefined value and null value

Undefined value means the variable is declared but not assigned.

Ex.

```
var x;  
console.log(x);  
x=10;
```

null value means the variable is declared and assigned as null means noting neighter number nor string or boolean.

```
var x=null;  
console.log(x);
```

Declaring are Assigning variables

```
var x=10; //Declare and Assigning
```

```
var t;    //Declaring  
var t=20; //Assigning
```

```
y=10; //Declaring auto and Assigning
```

"use strict"

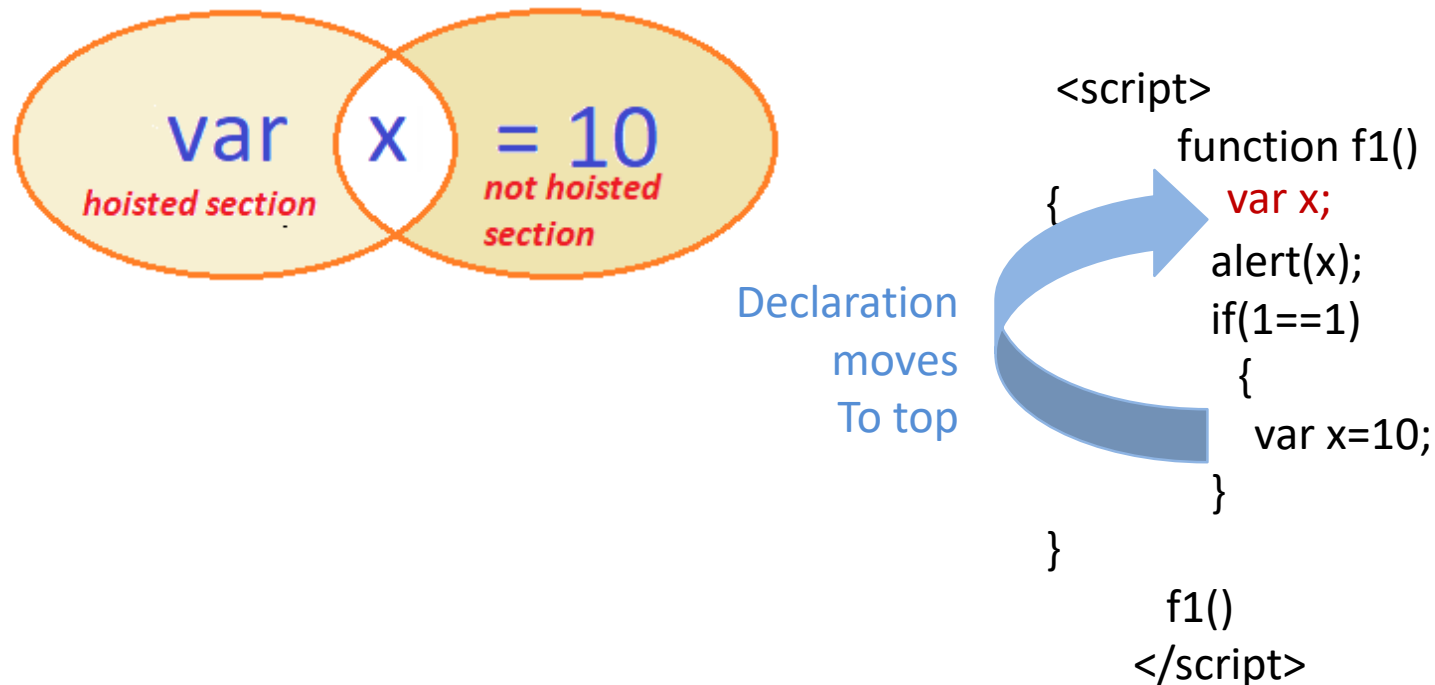
The strict mode in JavaScript does not allow following things:

- Use of undefined variables
- Use of reserved keywords as variable or function name
- Duplicate properties of an object
- Duplicate parameters of function
- Assign values to read-only properties
- Modifying arguments object

```
"use strict";  
x = 3.14;           // This will cause an error because x is not declared
```

JavaScript Hoisting

Hoisting is JavaScript's **default behavior of moving all declarations to the top of the current scope** (to the top of the **current script** or the **current function**).



let

`let` allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used. This is unlike the `var` keyword, which defines a variable globally, or locally to an entire function regardless of block scope.

```
<script>
{
  let x = 10;
  console.log(x);
}
console.log(x);
</script>
```

Javascript Operators

typeof: operator returns a string indicating the type of the unevaluated operand.

Syntax:

typeof operand

The following table summarizes the possible return values of typeof

Type	Result
Undefined	"undefined"
Null	"object"
Boolean	" <u>boolean</u> "
Number	"number"
String	"string"

Arithmetic Operator

Addition (+) Operator:

The addition operator produces the sum of numeric operands or string concatenation.

Type	Result
Number + Number (addition)	10 + 10 = 20
Boolean + Number (addition)	true + 10 = 11
Boolean + Boolean (addition)	true + true = 2 / false + false = 0
Boolean + Boolean + Number (addition)	true + true + 1 = 3 / true + false + 1 = 2
String + String (concatenation)	<u>"Infoway"</u> + <u>"Kothurd"</u> = <u>InfowayKothrud</u>
String + Number (concatenation)	<u>'Infoway'</u> + 10 = Infoway10
String + Boolean (concatenation)	"A" + true = Atrue
Number + String (concatenation)	10 + <u>"Infoway"</u> = 10Infoway
Number + String + Number (concatenation)	10 + <u>'Infoway'</u> + 10 = 10Infoway10
Number + Number + String (addition & concatenation)	10+10 + <u>"Infoway"</u> = 20Infoway
Boolean + Boolean + String (addition & concatenation)	true + true + <u>"Infoway"</u> = 2Infoway

Subtraction (-) Operator

Type	Result
Number - String	10 - 'A' = NaN
String - Number	"A" - 10 = NaN

Multiplication (*) Operator

Type	Result
Number * String	10 * 'A' = NaN
String * Number	"A" * 10 = NaN

Division (/) Operator

Type	Result
Number / String	10 / 'A' = NaN
Number / Number	10 / 0 = Infinity

Comparisons operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Equality (==) operators

The `equality (==)` operator converts the operands if they are not of the same type, then applies strict comparison.

Syntax:

`A==B`

```
1 == 1           // true
'1' == 1         // true
1 == '1'         // true
0 == false       // true
0 == null        // false
```

Identity / strict equality (===) operators

The `identity (===)` operator returns true if the operands are strictly equal with no type conversion.

Syntax:

`A===B`

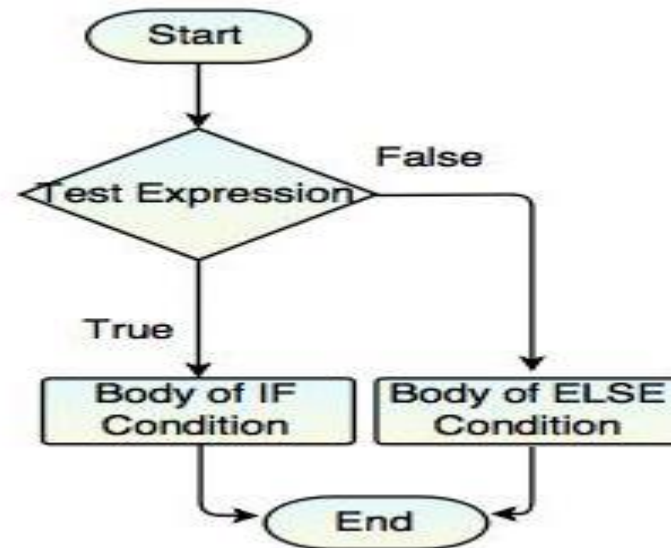
```
1 === 1      // true
```

```
1 === '1'    // false
```

if Statements

if-else statement is used to execute the code whether condition is true or false.

```
if(condition)
{
    //Statement 1;
}
Else
{
    //Statement 2;
}
```



Switch Statement

Switch is used to perform different actions on different conditions. It is used to compare the same expression to several different values.

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

Loops

for Loop:

```
for(initialization; test-condition;  
increment/decrement)  
{  
    //Statements;  
}
```

While Loop

```
while (condition)  
{  
    //Statements;  
}
```

Do-While Loop

```
do  
{  
    //Statements;  
}  
while(condition);
```

for...in loop is used to loop through an object's properties.

Events

Events are actions or occurrences that happen in the system you are programming, which the system tells you about so you can respond to them in some way if desired.

For example, if the user clicks a button on a webpage, you might want to respond to that action by displaying an information box.

```
<script type="text/javascript">  
  function click_event()  
  {  
    document.write("Button is clicked");  
  }  
</script>  
</head>  
<body>  
  <button onclick="click_event()">Click Me</button>  
</body>
```

Events	Description
onclick	occurs when element is clicked.
ondblclick	occurs when element is double-clicked.
onfocus	occurs when an element gets focus such as button, input, textarea etc.
onblur	occurs when form loses the focus from an element.
onsubmit	occurs when form is submitted.
onmouseover	occurs when mouse is moved over an element.
onmouseout	occurs when mouse is moved out from an element (after moved over).
onmousedown	occurs when mouse button is pressed over an element.
onmouseup	occurs when mouse is released from an element (after mouse is pressed).
onload	occurs when document, object or frameset is loaded.
onunload	occurs when body or frameset is unloaded.
onscroll	occurs when document is scrolled.
onresize	occurs when document is resized.
onreset	occurs when form is reset.
onkeydown	occurs when key is being pressed.
onkeypress	occurs when user presses the key.
onkeyup	occurs when key is released.

window.open()

Opens the new window.

syntax

```
var window = window.open(url, windowName, [windowFeatures]);
```

A DOM String indicating the URL of the resource to be loaded. This can be a path or URL to an HTML page, image file, or any other resource which is supported by the browser. If the empty string ("") is specified as url, a blank page is opened into the targeted browsing context.

```
<script>  
    function openwin(){  
        mywin=window.open("Page1.html","", "width=200,height=200,top=200,left=100");  
    }  
</script>
```

window.close()

closes the current window

syntax

```
window.close();
```

```
<script>  
    window.close();  
</script>
```

window.print()

prints the contents of the window.

syntax

```
window.print();
```

```
<script>  
    window.print();  
</script>
```

JavaScript Global Properties & methods

Infinity: A numeric value that represents positive/negative infinity

NaN: "Not-a-Number" value

eval(): Evaluates a string and executes it as if it was script code

isFinite(): Determines whether a value is a finite, legal number

isNaN(): Determines whether a value is an illegal number

Number(): Converts an object's value to a number

parseFloat(): Parses a string and returns a floating point number

parseInt(): Parses a string and returns an integer

END

Session-2

Topics to be covered...

- JavaScript Array
- JavaScript Array Methods
- JavaScript String
- String Methods
- JavaScript Dates Get/Set Methods
- JavaScript Math Object
- JavaScript RegEx

Array

JavaScript array is an object that represents a collection of similar type of elements.

Array indexes start with 0. [0] is the first array element, [1] is the second, [2] is the third ...

By array literal:

```
var arrayname=["value1","value2"....."valueN"];
```

By using an Array constructor (using new keyword):

```
var arrayname=new Array("value1","value2","value3");
```



```
<script>
var emp=new Array("one","two","three");
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

Output:

one
two
three

Array Properties

length property: The **length** property of an object which is an instance of type Array sets or returns the number of elements in that array.

Syntax: array.length

```
<script type="text/javascript">  
var arr = new Array( 10, 20, 30 );  
document.write("array length is : " + arr.length);  
</script>
```

Array-methods

concat(): Returns a new array comprised of this array joined with other array(s) and/or value(s).

Syntax:

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

toString(): Returns a string representing the array and its elements.

Syntax:

```
array.toString();
```

join(): Joins all elements of an array into a string.

It behaves just like toString(), but in addition we can specify the separator.

Syntax:

```
array.join(separator);
```

pop():

Removes the last element from an array and returns that element.

Syntax:

```
array.pop();
```

push():

Adds one or more elements to the end of an array and returns the new length of the array.

Syntax:

```
array.push(element1, ..., elementN);
```

shift(): Removes the first element from an array and returns that element.

Syntax:

```
array.shift();
```

unshift(): Adds one or more elements to the front of an array and returns the new length of the array.

Syntax:

```
array.unshift( element1, ..., elementN );
```

splice(): Adds and/or removes elements from an array.

Syntax:

```
array.splice(index, howMany, [element1][, ..., elementN]);
```

index – Index at which to start changing the array.

howMany – An integer indicating the number of old array elements to remove.

If **howMany** is 0, no elements are removed.

element1, ..., elementN – The elements to add to the array. If you don't specify any elements, splice simply removes the elements from the array.

slice(): Extracts a section of an array and returns a new array.

Syntax:

```
array.slice( begin [,end] );
```

It does not remove any elements from the source array.

sort() Sorts the elements of an array

Syntax:

```
array.sort( );
```

```
array.sort(compareFunction);
```

When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (*negative, zero, positive*) value

```
<script type="text/javascript">
  var num = [10,20,20,10];
  num.sort(function (a , b) {
    console.log(a +" & " + b + " " + (a-b));
  });
</script>
```

indexOf(): Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.

Syntax:

```
array.indexOf(searchElement[, fromIndex]);
```

lastIndexOf(): Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.

Syntax:

```
array.lastIndexOf(searchElement[, fromIndex]);
```

reverse(): Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.

Syntax:

```
array.reverse();
```

Array.forEach()

The forEach() method calls a function (a callback function) once for each array element.

Array.map()

The map() method creates a new array by performing a function on each array element.

The map() method does not execute the function for array elements without values.

The map() method does not change the original array.

Array.filter()

The filter() method creates a new array with array elements that passes a test.

Array.reduce()

The reduce() method reduces the array to a single value.

String Object

The String object is a constructor for strings, or a sequence of characters.

Syntax:

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

Length: Returns the length of the string.

Special Characters:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

String Methods

[charAt\(\)](#):Returns the character at the specified index

Syntax:

```
string.charAt(index);
```

index – An integer between 0 and 1 less than the length of the string.

Return Value

Returns the character from the specified index.

[charCodeAt\(\)](#)Returns a number indicating the Unicode value of the character at the given index.

Syntax:

```
string.charCodeAt(index);
```

[match\(\)](#):Used to match a regular expression against a string.

Syntax:

```
string.match( param )
```

param – A regular expression object.

[replace\(\)](#) Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.

Syntax:

```
string.replace(regex/substr, newSubStr/function[, flags]);
```

```
<script>
```

```
var re = "apples";
```

```
var str = "Apples are round, and apples are juicy.";
```

```
var newstr = str.replace(re, "oranges");
```

```
document.write(newstr );
```

```
</script>
```

To replace case insensitive, use a **regular expression** with an **/i** flag (insensitive):

To replace all matches, use a **regular expression** with a **/g** flag (global match)

search() Executes the search for a match between a regular expression and a specified string.

Syntax:

```
string.search(regex);
```

Str.toLowerCase(): Returns the calling string value converted to lower case.

Str.toUpperCase(): Returns the calling string value converted to uppercase.

endsWith(): Checks whether a string ends with specified string/characters

split() Splits a String object into an array of strings by separating the string into substrings.

Syntax:

```
string.split([separator][, limit]);
```

```
var x1 = ("Welcome to infoway technologies pune ".split());  
var x2 = (" Welcome to infoway technologies pune ".split(" "));  
var x3= (" Welcome to infoway technologies pune ".split(" ", 2));
```

substr() Returns the characters in a string beginning at the specified location through the specified number of characters.

Syntax:

```
string.substr(start[, length]);
```

```
document.write( str.substr(-2,2));
```

```
document.write(str.substr(1));
```

```
document.write( str.substr(-20,2));
```

substring() Returns the characters in a string between two indexes into the string.

Syntax:

```
string.substring(indexA, [indexB])
```

localeCompare(): Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.

Syntax:

```
string.localeCompare( param )
```

0 – If the string matches 100%.

1 – no match, and the parameter value comes before the *string* object's value in the locale sort order

-1 –no match, and the parameter value comes after the *string* object's value in the local sort order

Math object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

`Math.round(x)` returns the value of x rounded to its nearest integer

`Math.pow(x, y)` returns the value of x to the power of y

`Math.sqrt(x)` returns the square root of x

`Math.abs(x)` returns the absolute (positive) value of x

`Math.ceil(x)` returns the value of x rounded **up** to its nearest integer

`Math.floor(x)` returns the value of x rounded **down** to its nearest integer

`Math.min()` and `Math.max()` can be used to find the lowest or highest value in a list of arguments

Math.random() returns a random number between 0 (inclusive), and 1 (exclusive).

```
Math.floor(Math.random() * 10);    // returns a number between 0  
                                   and 9
```

```
Math.floor(Math.random() * 11);    // returns a number between 0  
                                   and 10
```

```
Math.floor(Math.random() * 100);   // returns a number between 0  
                                   and 99
```

```
Math.floor(Math.random() * 10) + 1; // returns a number between 1  
                                   and 10
```

```
Math.floor(Math.random() * 100) + 1; // returns a number between 1  
                                   and 100
```


Date Object

A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds.

Date objects are created with the **new Date()** constructor.

`new Date()`

`new Date(milliseconds)`

`new Date(dateString)`

`new Date(year, month, day, hours, minutes, seconds, milliseconds)`

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

```
<script>
var d = new Date();
document.write(d.getDay());
</script>
```

Date set Methods:

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

```
<script>
var d= new Date();
    d.setFullYear(2020, 0, 14);
    document.write(d);
</script>
```

Regular Expressions

Regular Expression Modifiers

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)

Quantifiers

Quantifier	Description
n+	Matches any string that contains at least one <i>n</i>
n*	Matches any string that contains zero or more occurrences of <i>n</i>
n?	Matches any string that contains zero or one occurrences of <i>n</i>

\$: Matches character at the end of the line.

. : Matches only period.

^ : Matches the beginning of a line or string.

- : Range indicator. [a-z, A-Z]

`[0-9]` : Find any character between the brackets (any digit)
`[^0-9]` : Find any character NOT between the brackets (any non-digit)
`[abc]` : Find any character between the brackets
`[^abc]` : Find any character NOT between the brackets
`(x|y)` Find any of the alternatives specified
`[a-z]` : It matches characters from lowercase 'a' to lowercase 'z'.
`[A-Z]` : It matches characters from uppercase 'A' to lowercase 'Z'.
`\w`: Matches a word character and underscore. (a-z, A-Z, 0-9, _).
`\W`: Matches a non word character (% , # , @ , !).
`\d`: Find a digit
`\s`: Find a whitespace character
`{M, N}` : Donates minimum M and maximum N value.

RegEx Object Methods:

`exec()`: Tests for a match in a string. Returns the first match

`test()`: Tests for a match in a string. Returns true or false

END

Session-3 & 4

Topics to be covered...

- DOM Methods
- DOM Document
- DOM Elements
- DOM HTML
- DOM CSS
- DOM Animations
- DOM Event Listener
- JavaScript Form Validation
- DOM Nodes

HTML DOM

The **document object** represents the whole html document.

When a webpage loads in the browser. browser creates a DOM for the page.

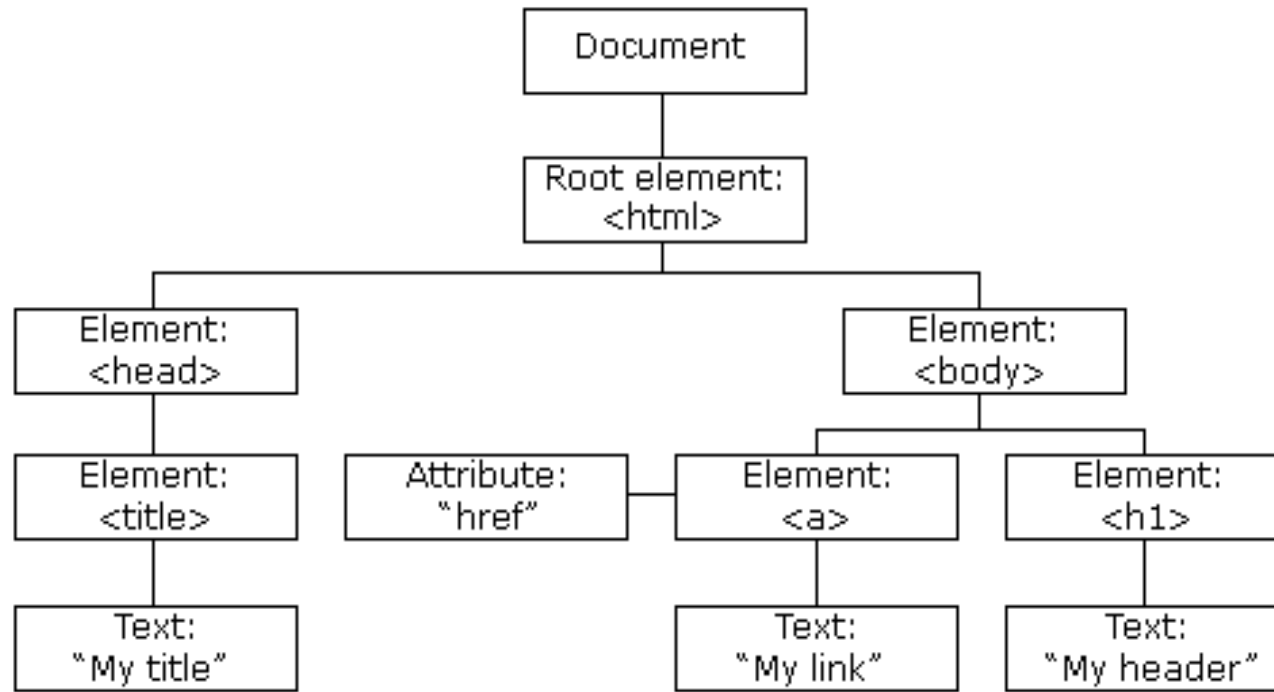
Document:- Html Page

Object:- Elements and attributes

Model:- Tree Structure of HTML elements

By the help of document object, we can add dynamic content to our web page.

The DOM is a W3C (World Wide Web Consortium) standard.



Why we should learn HTML DOM: w3 standard to understand the structure of html page so that we can create ,read, update and delete and manage DOM elements using JavaScript and jQuery methods.

HTML DOM helps you to understand and control the element structure using JavaScript methods.

DOM Methods

document.write("string")

writes the given string on the document.

document.getElementById()

returns the element having the given id value.

getElementsByTagName()

returns all the elements having the given name value.

getElementsByTagName()

returns all the elements having the given tag name.

getElementsByClassName()

returns all the elements having the given class name

innerHTML Property

The **innerHTML** property can be used to write the dynamic html on the html document.

Syntax:

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

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

```
<p id="para">Old text!</p>
```

innerText Property

The innerText property sets or returns the text content of the specified node.

Syntax:

```
document.getElementById(id).innerText = text
```

```
<script>  
document.getElementById("para").innerText = "New text!";  
</script>
```

```
<p id="para">Old text!</p>
```

DOM createElement()

The createElement() method creates an Element Node with the specified name.

syntax

```
var element = document.createElement(tagName);
```

```
var para= document.createElement("p");
```

DOM createTextNode()

To add text to the element we have to create a textnode first.

syntax

```
var node= document.createTextNode("text");
```

```
var node = document.createTextNode("This is a new  
paragraph.");
```

HTML DOM textContent Property

The textContent property sets or returns the textual content of the specified node.

syntax

`node.textContent = text`

```
Para.textContent="This is a new paragraph.";
```

HTML DOM setAttribute() Method

The setAttribute() method adds the specified attribute to an element, and gives it the specified value.

syntax

```
element.setAttribute(attributename, attributevalue)
```


DOM Node.appendChild()

The Node.appendChild() method adds a node to the end of the list of children of a specified parent node.

syntax

Node.appendChild()

```
<div id="div1">  
</div>
```

```
<script>  
var para = document.createElement("p");  
var node =  
document.createTextNode("Infoway");  
para.appendChild(node);
```

```
var element =  
document.getElementById("div1");  
element.appendChild(para);  
</script>
```

HTML DOM EventListener

The `addEventListener()` method attaches an event handler to the specified element.

You can add many event handlers to one element

Syntax:

```
element.addEventListener(event, function);
```

```
<button id="myBtn">Handler</button>
```

```
<script>
```

```
document.getElementById("myBtn").addEventListener("click",  
myFunction);
```

```
function myFunction() {  
    alert ("Hello World!");  
}  
</script>
```

Event Bubbling or Event Capturing?

Event propagation is a way of defining the element order when an event occurs. If you have a `<p>` element inside a `<div>` element, and the user clicks on the `<p>` element, which element's "click" event should be handled first?

In *bubbling* the inner most element's event is handled first and then the outer: the `<p>` element's click event is handled first, then the `<div>` element's click event.

In *capturing* the outer most element's event is handled first and then the inner: the `<div>` element's click event will be handled first, then the `<p>` element's click event.

Syntax:

```
addEventListener(event, function, useCapture)
```

default value is false which will use the bubbling propagation,

END

Session-5

Topics to be covered...

- Canvas Methods
- JavaScript Window
- JavaScript Screen
- JavaScript Location
- JavaScript History
- JavaScript Navigator
- JavaScript Timing
 - setTimeout()
 - setInterval()
- JavaScript Objects
- Properties and Methods

Canvas

- a. The HTML <canvas> element is used to draw graphics on a web page.
- b. Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

Step 1: Find the Canvas Element

```
var canvas = document.getElementById("myCanvas");
```

Step 2: Create a Drawing Object

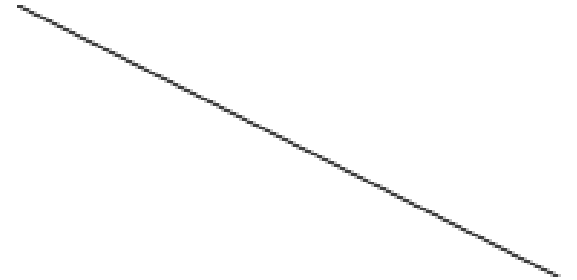
```
var ctx = canvas.getContext("2d");
```

Draw a Line:

moveTo(x,y) - defines the starting point of the line

lineTo(x,y) - defines the ending point of the line

```
var canvas = document.getElementById("CN");  
var ctx = canvas.getContext("2d");  
ctx.moveTo(0, 0);  
ctx.lineTo(200, 100);  
ctx.stroke();
```

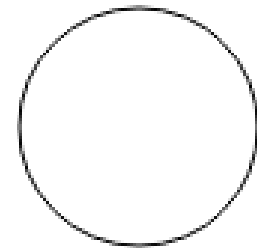


Draw a Circle:

beginPath() - begins a path

arc(x,y,r,startangle,endangle) - creates an arc/curve.

```
var canvas = document.getElementById("CN");  
var ctx = canvas.getContext("2d");  
ctx.beginPath();  
ctx.arc(95, 50, 10, 0, 2 * Math.PI);  
ctx.stroke();
```



Drawing Text on the Canvas:

```
var canvas = document.getElementById("CN");  
var ctx = canvas.getContext("2d");  
ctx.font = "30px Arial";  
ctx.fillText("Welcome", 10, 50);
```

A rectangular canvas with a thin black border. Inside, the word "Welcome" is written in a black, sans-serif font, positioned towards the top-left corner.

Welcome

Add Color and Center Text:

```
var canvas = document.getElementById("myCanvas");  
var ctx = canvas.getContext("2d");  
ctx.font = "30px Comic Sans MS";  
ctx.fillStyle = "red";  
ctx.textAlign = "center";  
ctx.fillText("Welcome", 50, 50);
```

A rectangular canvas with a thin black border. Inside, the word "Welcome" is written in a red, rounded, sans-serif font, centered horizontally and vertically.

Welcome

Window location object

The `window.location` object can be used to get the current page address (URL) and to redirect the browser to a new page.

`window.location.href` returns the href (URL) of the current page

`window.location.hostname` returns the domain name of the web host

`window.location.pathname` returns the path and filename of the current page

`window.location.protocol` returns the web protocol used (http: or https:)

`window.location.port` property returns the number of the internet host port (of the current page).

Window History object

The window.history object contains the browsers history.

history.back() - same as clicking back in the browser

history.forward() - same as clicking forward in the browser

history.go() - loads the given page number.

```
history.back();    //for previous page
history.forward(); //for next page
history.go(2);     //for next 2nd page
history.go(-2);    //for previous 2nd page
```

Window navigator object

The **JavaScript navigator object** is used for browser detection. It can be used to get browser information such as appName, appCodeName etc.

appName	returns the name
appVersion	returns the version
appCodeName	returns the code name
cookieEnabled	returns true if cookie is enabled otherwise false
language	returns the language
platform	returns the platform e.g. Win32.
online	returns true if browser is online otherwise false..

JS Timer

Window.setTimeout() Method:

Executes a function, after waiting a specified number of milliseconds.

Syntax:

```
setTimeout(function, milliseconds);
```

The first parameter is a function to be executed.
The second parameter indicates the number of milliseconds before execution.

```
<button onclick="setTimeout(myFunction,  
3000)">SetTimeout</button>
```

```
<script>  
function myFunction() {  
    alert('Hello');  
}  
</script>
```

Window.setInterval() Method

The setInterval() method repeats a given function at every given time-interval.

Syntax:

```
setInterval(function, milliseconds);
```

The first parameter is the function to be executed. The second parameter indicates the length of the time-interval between each execution.

```
<script>
var myVar = setInterval(starttime, 1000);

function starttime() {
    var d = new Date();
    document.write(d.toLocaleTimeString());
}
</script>
```

clearInterval() method:

The clearInterval() method stops the executions of the function specified in the setInterval() method.

Syntax:

```
window.clearInterval(timerVariable)
```

```
<p id="starttime"></p>
```

```
<button onclick="clearInterval(stime)">Stop time</button>
```

```
<script>
```

```
var stime= setInterval(stopWatch, 1000);
```

```
function stopWatch() {
```

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


```
    document.getElementById("starttime").innerHTML =  
d.toLocaleTimeString();
```

```
}
```

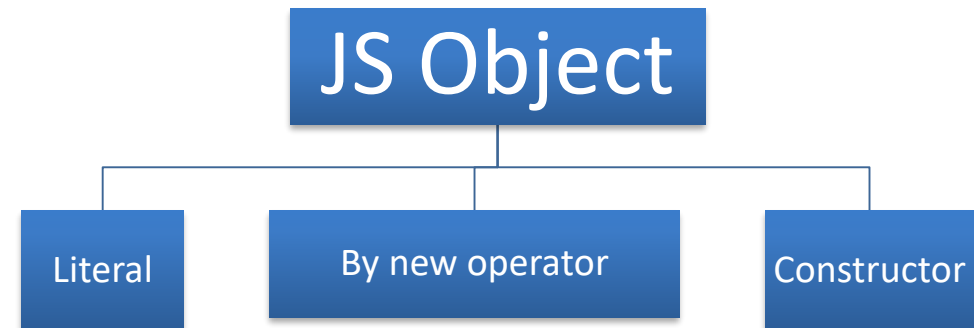
```
</script>
```

Javascript Object

A javascript object is an entity having state and behavior (properties and method).

Object	Properties	Methods
	<code>car.name = Fiat</code> <code>car.model = 500</code> <code>car.weight = 850kg</code> <code>car.color = white</code>	<code>car.start()</code> <code>car.drive()</code> <code>car.brake()</code> <code>car.stop()</code>

Creating Objects in JavaScript:



JavaScript Object by object literal

Syntax:

```
object={property1:value1,  
        property2:value2.....propertyN:valueN}
```

```
Var employee=  
{  
  name:"Ram Kumar",  
  salary:40000  
}  
document.write(employee.name+" "+employee.salary);
```

method in JavaScript Object literal

```
<p id="demo"></p>
```

```
<script>
```

```
var person = {
```

```
    firstName: "Sohan",
```

```
    lastName : "Lal",
```

```
};
```

```
person.name = function() {
```

```
    return this.firstName + " " + this.lastName;
```

```
};
```

```
document.getElementById("demo").innerHTML ="Name is " +
```

```
person.name();
```

```
</script>
```

For...in loop

Syntax:

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

The block of code inside of the for...in loop will be executed once for each property.

```
<script>  
var p;  
var person = {fname:"Ram", lname:"Sharma", age:25};  
  
for (x in person) {  
    p += person[x];  
}  
Document.write(p);  
</script>
```

By The new Operator

All user-defined objects and built-in objects are descendants of an object called **Object**.

Syntax:

```
var objectname=new Object();
```

```
var employee=new Object();
```

```
employee.id=101;
```

```
employee.name="Ram Sharma";
```

```
employee.salary=50000;
```

```
document.write(employee.id+" "+employee.name+" "+employee.salary);
```

Defining method

```
<script>
  var Person = new Object();
  Person.id = 1001;
  Person['n'] = 'Infoway Technologies';
  Person.getName = function () {
    return (Person.n);
  }
  alert(Person.getName());
</script>
```

By using an Object constructor

We need to create function with arguments. Each argument value can be assigned in the current object by using this keyword. The **this keyword** refers to the current object.

```
function employee(){  
  this.name="Smith";  
  this.salary=50000;  
}  
  
var e=new employee();  
Document.write(e.name+"  
"+e.salary);
```

```
<p id="demo"></p>
<script>
// Constructor function for Person objects
function Person(first, last, age) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.name = function() {
        return this.firstName + " " + this.lastName
    };
}

// Create a Person object
var p= new Person("Sohan", "Lal", 50);

// Display full name
document.getElementById("demo").innerHTML = p.name();

</script>
```

Object Properties

An **object** is a collection of properties, and a property is an association between a **name** (or **key**) and a **value**. A property's value can be a **function**, in which case the property is known as a **method**.

syntax

- `objectName.property` *// person.age*
- `objectName["property"]` *// person["age"]*

```
<script type="text/javascript">
  var Person = new Object();
  Person.id = 1001;
  Person['n'] = 'Infoway Technologies';
  alert(Person.n);
</script>
```


Deleting Properties

The **delete** keyword deletes a property from an object.

The delete keyword deletes both the value of the property and the property itself.

It has no effect on variables or functions.

The delete operator should not be used on predefined JavaScript object properties.

```
<script type="text/javascript">  
  var Person = new Object();  
  Person.id = 1001;  
  delete Person.id;  
  console.log(Person.id);  
</script>
```

JavaScript Object Prototypes

All JavaScript objects inherit properties and methods from a prototype.

Prototype Inheritance:

The `Object.prototype` is on the top of the prototype inheritance chain.
Date objects, Array objects, and Person objects inherit from `Object.prototype`.

Adding Properties and Methods to Objects

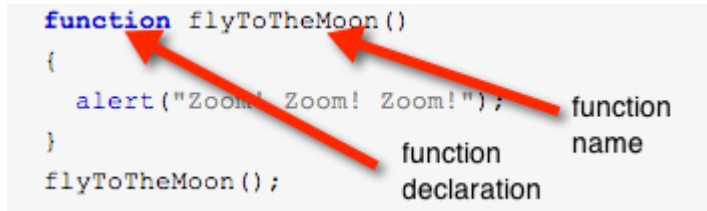
```
Person.prototype.course = "PG-DAC";
```

```
Person.prototype.name = function() {  
    return this.fname + " " + this.lname;  
};  
document.write(p.name());
```

Function Expressions

A JavaScript function can also be defined using an **expression**.

```
var x = function (a, b) {return a * b}; //anonymous function (a function without a name).  
var z = x(4, 3);
```



```
function flyToTheMoon()  
{  
    alert("Zoom! Zoom! Zoom!");  
}  
flyToTheMoon();
```

Annotations for the function declaration:

- function**: points to the keyword
- flyToTheMoon()**: points to the function name and its parameters
- function name**: points to the name `flyToTheMoon()`
- function declaration**: points to the entire function block



```
var flyToTheMoon = function(  
{  
    alert("Zoom! Zoom! Zoom!");  
})
```

Annotations for the function expression:

- var**: points to the keyword
- flyToTheMoon**: points to the variable name
- =**: points to the assignment operator
- function**: points to the keyword
- function operator**: points to the `= function` sequence
- variable that stores returned object**: points to the variable `flyToTheMoon`
- function body**: points to the code inside the curly braces

Function Hoisting:

```
myFunction(5);
```

```
function myFunction(y) {  
    return y * y;  
}
```

Functions defined using an expression are not hoisted

Self-Invoking Functions

A self-invoking (also called self-executing) function is a nameless (anonymous) function that is invoked immediately after its definition.

An anonymous function is enclosed inside a set of parentheses followed by another set of parentheses (), which does the execution.

```
(function() {  
    ...  
    ...  
}) ();
```

Self-invoking functions are useful for initialization tasks and for one-time code executions, without the need of creating global variables.

```
(function(){  
    console.log(Math.PI);  
})();
```

```
(function(x){  
    console.log(x);  
})("Hello, World!");
```

Function Closure

Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.

Inner function can access variables and parameters of an outer function (however, cannot access arguments object of outer function).

```
var incrementClickCount = (function ()  
{  
    var clickCount = 0;  
    return function ()  
    {  
        return ++clickCount;  
    }  
})();
```

```
<input type="button" value="Click Me" onclick="alert(incrementClickCount());" />
```

When to use Closure?

Closure is useful in hiding implementation detail in JavaScript. In other words, it can be useful to create private variables or functions.

END

Session-6 & 7

Topics to be covered...

- Introduction to jQuery
- Why we use jQuery
- Ready function
- jQuery Selectors
- jQuery HTML
- jQuery Effects
- jQuery Events
- jQuery Form Validation
- jQuery UI

Introduction to jQuery

jQuery is a lightweight JavaScript library that emphasizes interaction between JavaScript and HTML

It is cross-platform and supports different types of browsers

Developed in 2006 by John Resig at Rochester Institute of Technology

What can jQuery Do

jQuery

Select
elements
by
tag,class
,Id

Handle
events in
same way
across
browsers

Supports
for
animation

Eases
service
and Ajax
calls

helps in
applying
styles to
multiple
elements

Why jQuery

jQuery is like a pre-packaged set of JavaScript routines that you may have otherwise needed to write yourself, packaged in an easy-to-use way.

It uses CSS syntax for selection of element.

jQuery makes animated applications just like Flash

jQuery pages load faster

jQuery vs. JavaScript

- Example 1 - Hide an element with id "textbox"

//javascript

```
document.getElementById('textbox').style.display = "none";
```

//jQuery

```
$('#textbox').hide();
```

- Example 2 - Create a <h1> tag with "my text"

//javascript

```
var h1 = document.createElement("h1");
```

```
h1.innerHTML = "my text";
```

```
document.getElementsByTagName('body')[0].appendChild(h1);
```

//jQuery

```
$('body').append( $("

# 

").html("my text") );
```

Getting started with jQuery

Two ways to access it:

-Download it and reference:

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

(in the same directory)

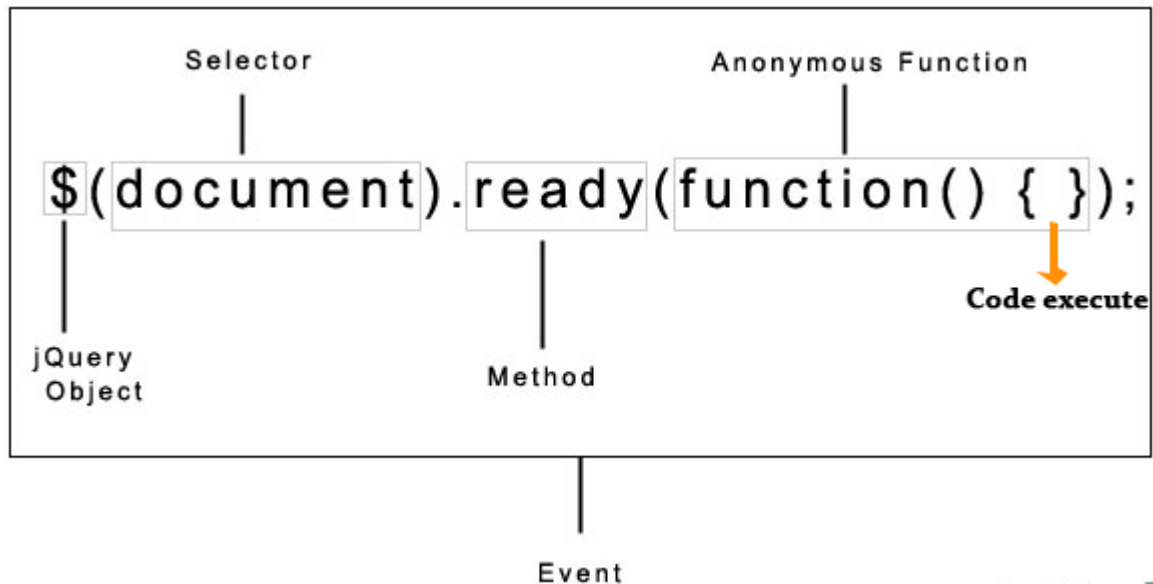
-Reference the JS file in your HTML (CDN):

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.js">  
</script>
```

`$(document).ready()`

A page can't be manipulated safely until the document is "ready."

jQuery detects this state of readiness for you. Code included inside `$(document).ready()` will only run once the page Document Object Model is ready for JavaScript code to execute.



Basic Selectors

jQuery selectors allow you to select and manipulate HTML element(s).

- TagName

```
document.getElementsByTagName("tagName");  
$("tagName") - $("div"), $("p"), $("div"),.....
```

- Tag ID

```
document.getElementById("id");  
$("#id") - $("#name"), $("#address")
```

- Tag Class

```
document.getElementsByClassName("className");  
$(".className") - $(".comment"), $(".code")
```

- To select all elements - \$("*")

Selectors - Combined

`$("tagName,.className")`

Example: `$("h1,.mainTitle")`

`$("tagName,.className,#tagId")`

Example: `$("h1,.mainTitle,#firstHeading")`

`$("selector:first")`

`$("selector:last")`

`$("selector:odd")`

`$("selector:even")`

`$("selector[attribute]")`

Examples:

`$("p:first")`

`$("p:last")`

`$("p:odd")`

`$("p:even")`

`$("p:[name]")`

Condition filters - Form filters

`$("selector:visible")`

`$("selector:disabled")`

`$("selector:enabled")`

`$("selector:checked")`

`$("selector:button")`

`$("selector:file")`

`$("selector:input")`

`$("selector:text")`

`$("selector:password")`

`$("selector:checkbox")`

`$("selector:submit")`

`$("selector:reset")`

:checkbox selector

```
<script>
$(document).ready(function(){
$("button").click(function(){
    $("input:checkbox").each(function(k,v){
        alert(k+v+v.value);
    }); }); });
</script>
```

```
$(document).ready(function(){
    $("input:checkbox").click(function () {
        var x = "";
        $("#span1").text("");
        $("input:checked").each(function (k, v) {
            x += v.value;
        });
        $("#span1").text(x);
    })
});
</script>
```

Retrieve, Set and Remove attributes

`$("selector").attr("name")`

`$("selector").attr("key", "val")`

`$("selector").attr(properties)`

`$("selector").removeAttr(attr)`

Examples:

`$("img").attr("src")`

`$("p").attr("class", "source")`

`$("img").attr({ "src" : "/path/",
"title" : "My Img" });`

`$("div").removeAttr("class")`

Class, HTML, Text, Value - Functions

```
$("#selector").addClass("className")
```

```
$("#selector").removeClass("className")
```

```
$("#selector").toggleClass("className")
```

html() - Sets or returns the content of selected elements (including HTML markup)

```
$("#selector").html()
```

```
$("#selector").html("html code")
```

text() - Sets or returns the text content of selected elements

```
$("#selector").text()
```

```
$("#selector").text("text content")
```

val() - Sets or returns the value of form fields

```
$("#selector").val()
```

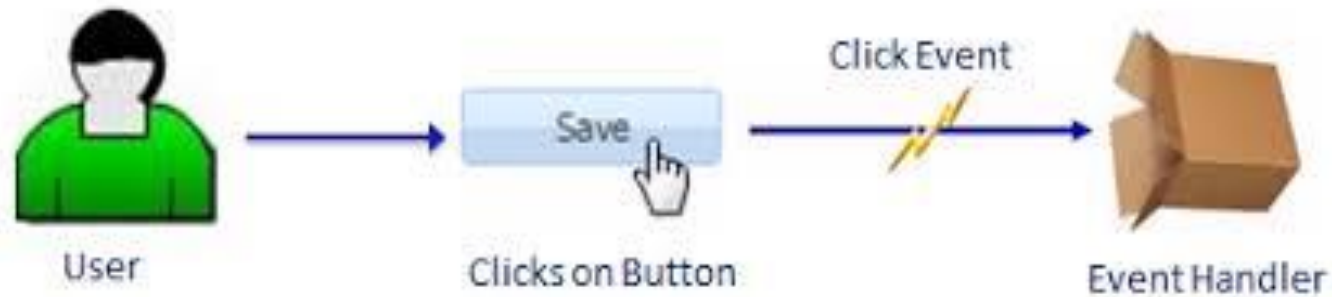
```
$("#selector").val("value")
```

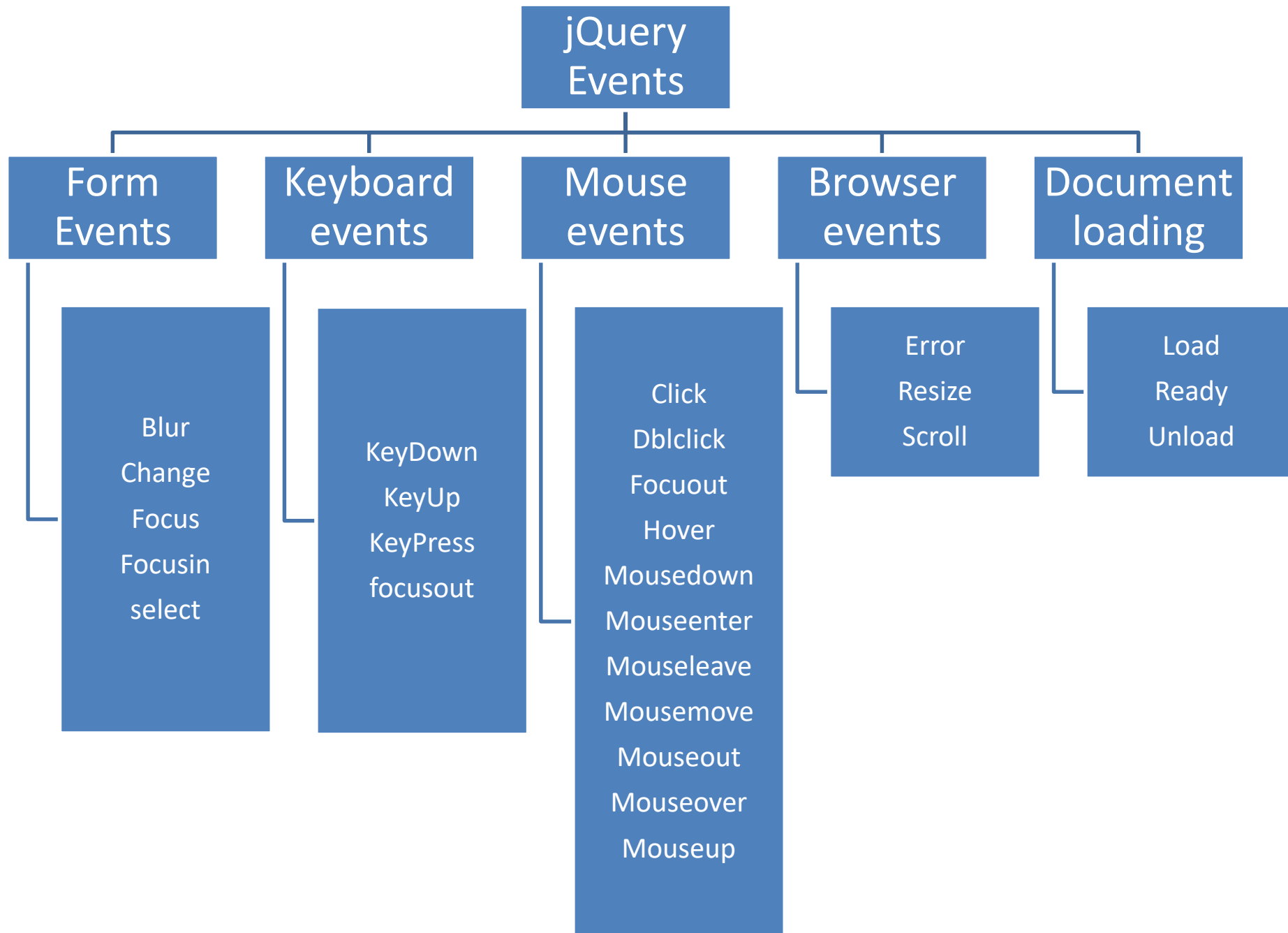
```
<script>
    $(document).ready(function () {
        $("#button1").click(function () {
            alert($("#p1").text());
            alert($("#div1").html());
            alert($("#text1").val());
        });
    });
</script>
```

```
<script>
    $(document).ready(function () {
        $("#btn1").click(function(){
            $("#p1").text("Hello world!");
        });
        $("#btn2").click(function(){
            $("#p2").html("<b>Welcome</b>");
        });
        $("#btn3").click(function(){
            $("#text1").val("Infoway");
        });
    });
</script>
```

jQuery Events

Events are a part of the Document Object Model (DOM) and every HTML element contains a set of events which can trigger JavaScript Code.





```
// Event setup using a convenience method  
$( "p" ).click(function() {  
  console.log( "You clicked a paragraph!" );  
});
```

```
// Equivalent event setup using the `.on()` method  
$( "p" ).on( "click", function() {  
  console.log( "You clicked a paragraph!" );  
});
```

Keydown event

```
var x = event.which || event.keyCode; //
```

Use either which or keyCode, depending on browser support.

```
<script>
  $(document).ready(function () {
    $("#text1").keydown(function () {
      $("#text2").val(event.which);
    });
  });
</script>
```

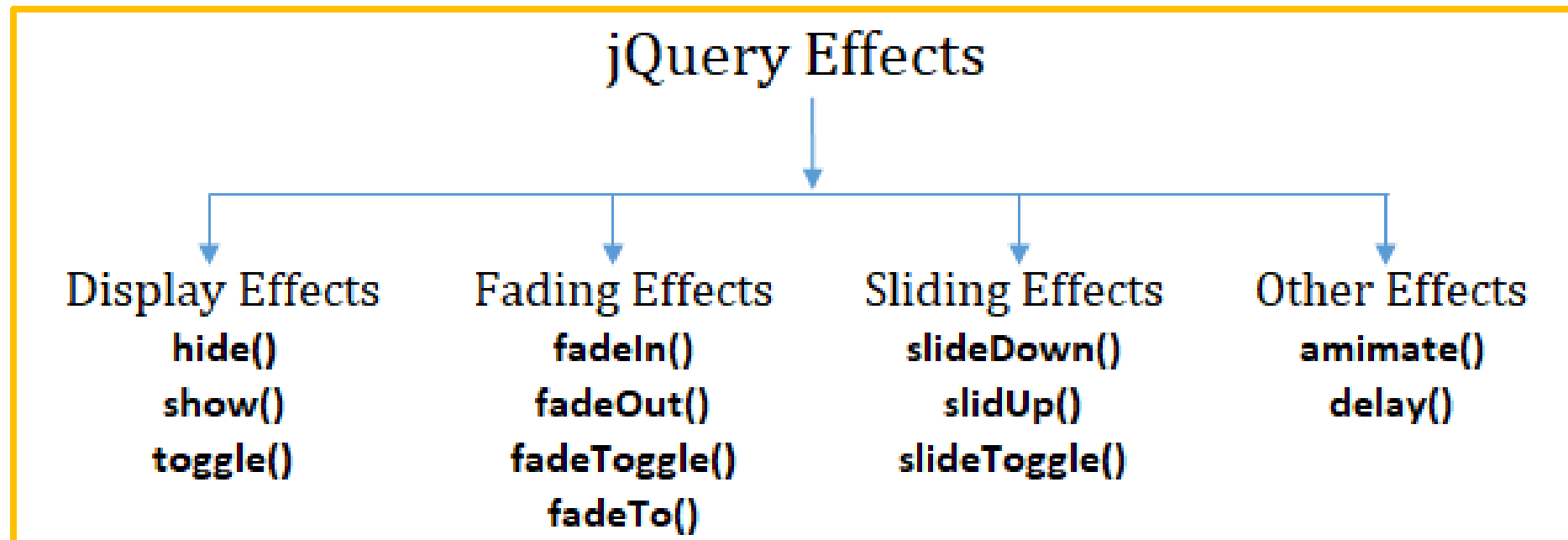
Focusout event

The focusout event is sent to an element when it, or any element inside of it, loses focus. Bind an event handler to the "focusout" event.

```
<script>
  $(document).ready(function () {
    $(":text").focusin(function () {
      $(this).css("background-color", "red");
    });
    $(":text").focusout(function () {
      $(this).css("background-color", "white");
    });
  });
</script>
```

jQuery Effects

The jQuery library provides several techniques for adding animation to a web page. These include simple, standard animations that are frequently used, and the ability to craft sophisticated custom effects.



Display Effects - .hide() and .show()

Hide the matched elements.

Show the matched elements

Syntax:

```
$(selector).hide(speed,callback);
```

```
$(selector).show(speed,callback);
```

Speed is given in milliseconds; higher values indicate slower animations, not faster ones. The strings 'fast' and 'slow' can be supplied to indicate durations of 200 and 600 milliseconds, respectively.

jQuery toggle():

Syntax:

```
$(selector).toggle(speed,callback);
```

Fading Effects - `.fadeIn()`, `.fadeOut()`

Display the matched elements by fading them to opaque.

Hide the matched elements by fading them to transparent.

syntax

```
$(selector).fadeIn(speed,callback);
```

```
$(selector).fadeOut(speed,callback);
```

Speed is given in milliseconds; higher values indicate slower animations, not faster ones. The strings 'fast' and 'slow' can be supplied to indicate durations of 200 and 600 milliseconds, respectively. If any other string is supplied, or if the duration parameter is omitted, the default duration of 400 milliseconds is used.

Fading Effects - .fadeTo(), .fadeToggle()

Adjust the opacity of the matched elements.

Display or hide the matched elements by animating their opacity.

opacity: A number between 0 and 1 denoting the target opacity.

Syntax:

```
$(selector).fadeTo(speed,opacity,callback);
```

```
$(selector).fadeToggle(speed,callback);
```

```
<script>
$(document).ready(function () {
  $("#toggle").on("click", function () {
    $("#div1").fadeTo("slow",0.5, function () {
      alert("Hello World!");
    });
  });
});
</script>
```


Sliding Effects - `.slideDown()`, `.slideUp()`

Display the matched elements with a sliding motion.

Hide the matched elements with a sliding motion.

Syntax:

```
$(selector).slideDown(speed,callback);
```

```
$(selector).slideUp(speed,callback);
```

```
$(selector).slideToggle(speed,callback);
```

Stop()

The jQuery stop() method is used to stop animations or effects before it is finished.

Syntax:

```
$(selector).stop();
```

Other Effects - .animate()

Perform a custom animation of a set of CSS properties.

Syntax:

`$(selector).animate({params},speed,callback);`

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!

HTML

```
<button id="left">left</button>
<button id="right">right</button>
<div class="block"></div>
```

JS

```
$(document).ready(function(){
  $(".block").css({
    'position': 'absolute',
    'backgroundColor': "#abc",
    'left': '100px',
    'width': '90px',
    'height': '90px',
    'margin': '5px' });
  $("#left").click(function(){
    $(".block").animate({left: "-=50px"}, "slow");
  });
  $("#right").click(function(){
    $(".block").animate({left: "+=50px"}, "slow");
  });
});
```

jQuery-CSS

These methods get and set CSS-related properties of elements.

Set a CSS Property:

Syntax:

```
css("propertyname", "value");
```

```
$("#p").css("background-color", "yellow");
```

jQuery - Add Elements

- `append()` - Inserts content at the end of the selected elements
- `prepend()` - Inserts content at the beginning of the selected elements
- `after()` - Inserts content after the selected elements
- `before()` - Inserts content before the selected elements

jQuery - Remove Elements

- `remove()` - Removes the selected element (and its child elements)
- `empty()` - Removes the child elements from the selected element

jQuery - Widgets

A jQuery UI widget is a specialized jQuery plug-in. Using plug-in, we can apply behaviours to the elements.

Autocomplete

Enable to provides the suggestions while you type into the field.

Menu

Menu shows list of items.

Datepicker

It is to open an interactive calendar in a small overlay.

Select menu

Enable a style able select element/elements.

Tooltip

Its provides the tips for the users.

Tabs

It is used to swap between content that is broken into logical sections.

END

ECMA Script 6

What is ES6?

The most recent version of ECMAScript.

ES1: june 1997

ES2: june 1998

ES3: December 1999

ES4: Abandoned

ES5: december 2009

ES6/ES2015:june 2015

ES7/ES2016:june 2016

ES8/ES2017:june 2017

ES.Next: this term is dynamic and references the next version of ECMAScript coming out.

Transpilers

Compilers translate code one language to another ex. Java to bytecode Transpilers translate code to the same language There are several transpilers that translate ES6 code to ES5

ES6 Transpilers

Traceur - 64% from Google generates source maps doesn't work with IE8 and below due to use of ES5 get/set syntax
<https://github.com/google/traceur-compiler/>

Babel - 76% aims to generate ES5 code that is as close as possible to the input ES6 code generates source maps some features don't work with IE10 and below see
<https://babeljs.io/docs/usage/caveats/#internet-explorer>
<https://babeljs.io>

TypeScript - 9% from Microsoft "a typed superset of JavaScript that compiles to plain JavaScript. Any browser. Any host. Any OS. Open Source." supports optional type specifications for variables, function return values, and function parameters has goal to support all of ES6 generates source maps to install, npm install -g typescript to compile, tsc some-file.ts generates some-file.js <http://www.typescriptlang.org>

Goals of ES6

Be taken more seriously

Fix some issues from ES5

Backward compatibility –ES5 should work in ES6

Modern Syntax

Better Scaling & Better for Big Applications

New Features in Standard Library

What 's New in ES6?

Let and const declarations

Default parameter

Rest operator

Spread operator

Destructing Assignment

Arrow functions

Classes

Symbol

Template String

for-of loop

Promises

Modules (import and export)

collection (Map and Set)

new methods in String and Object classes

iterators

Generators

Var vs. let vs. const

Var is function scoped.

Let and const are block scoped

Let is not hoisted to beginning of enclosing block, so references before declaration are errors
most uses of var can be replaced with let (not if they depend on hoisting)

const declares constants with block scope

must be initialized

reference can't be modified, but object values can

to prevent changes to object values, use `Object.freeze(obj)`

let

```
1 for (let i = 0; i < 10; ++i) {  
2   console.log(i); // 0, 1, 2, 3, 4 ... 9  
3 }  
4  
5 console.log(i); // i is not defined
```

const

```
1 const PI = 3.14159265359;  
2 PI = 0; // => 0  
3 console.log(PI); // => 3.14159265359
```

Let and const variables are hoisted but they are not initialized yet unlike vars and functions.

But, What should I use? var?let?const?

The only difference between const and let is that const makes the contract that no rebinding will happen.

Use const by default.

Only use let if rebinding is needed.

Var shouldn't be used in ES2015.

Use var for top level variables

Use let for localized variable in smaller scope

Refactor let to const only after some code has been written and you're reasonably sure there shouldn't be variable reassignment.

Modules

Modules allow you to load code asynchronously and provides a layer of abstraction to your code.

Exports

```
1 //lib/math.js
2
3 export function sum(x, y) {
4   return x + y;
5 }
6
7 export var pi = 3.141593;
```

Imports

```
1 // app.js
2
3 import {sum, pi} from "lib/math";
4
5 alert("2π = " + sum(pi, pi));
```

Arrow Functions

More concise than traditional function expressions.

Syntax:

(params) => { expressions }

```
// Traditional function expression.  
const addNumbers = function (num1, num2) {  
    return num1 + num2;  
}  
  
// Arrow function expression with implicit return.  
const addNumbers = (num1, num2) => num1 + num2;
```

Few more examples:

```
// Arrow function without any arguments.  
const sayHello = () => console.log( 'Hello!' );  
  
// Arrow function with a single argument.  
const sayHello = name => console.log( 'Hello ${name}!' );  
  
// Arrow function with multiple arguments.  
const sayHello = (fName, lName) => console.log( 'Hello ${fName} ${lName}!' );
```

Default Parameter

```
function f(x = 0, y = 0) {  
  }  
f(); // 0,0  
f(5); // 5,0  
f(5, someVarUndefined); // 5,0  
f(undefined, 6); // 0, 6  
f(5,6); // 5,6  
Function(x, y=x) { } //OK  
Function (x=y, y=5) { } // y is not available
```

...Rest Operator

The rest operator allows us to more easily handle a variable number of function parameters.

```
function doMath(operator, ...numbers) {  
  console.log(operator); // 'add'  
  console.log(numbers); // [1, 2, 3]  
}  
  
doMath('add', 1, 2, 3);
```

...Spread Operator

Before ES6 we would run `.apply()` to pass in an array of arguments.

```
function doSomething (x, y, z) {  
    console.log(x, y, z);  
}  
let args = [0, 1, 2];  
  
// Call the function, passing args.  
doSomething.apply(null, args);
```

But with ES6, we can use the spread operator `...` to pass in the arguments.

```
function doSomething (x, y, z) {  
    console.log(x, y, z);  
}  
let args = [0, 1, 2];  
  
// Call the function, without 'apply', passing args with the spread operator!  
doSomething(...args);
```

```
let array1 = ['one', 'two', 'three'];  
let array2 = ['four', 'five'];  
  
array1.push(...array2) // Adds array2 items to end of array  
array1.unshift(...array2) // Adds array2 items to beginning of array
```

```
let array1 = ['two', 'three'];  
let array2 = ['one', ...array1, 'four', 'five'];  
  
console.log(array2); // ["one", "two", "three", "four", "five"]
```

```
let array1 = [1,2,3];  
let array2 = [...array1]; // like array1.slice()  
array2.push(4)  
  
console.log(array1); // [1,2,3]  
console.log(array2); // [1,2,3,4]
```

Destructuring

Assigns values to any number of variables from values in arrays and objects

Destructuring Arrays

```
const details = [ 'Kevin', 'Langley', 'kevinlangleyjr.com' ];  
  
const [ first, last, website ] = details;  
  
console.log(first); // Kevin  
console.log(last); // Langley  
console.log(website); // kevinlangleyjr.com
```

Destructuring Object

```
const person = {  
  first: 'Kevin',  
  last: 'Langley',  
  location: {  
    city: 'Beverly Hills',  
    state: 'Florida'  
  }  
};  
  
const { first, last } = person;
```

It even works with nested properties:

```
const { city, state } = person.location;
```

You can also rename the variables from the destructuring object:

```
const { first: fName, last: lName } = person;  
const { city: locationCity, state: locationState } = person.location;
```

Template Literals

Template Literal is a new way to create a string

```
const name = 'Kevin';

// The old way...
console.log('Hello, ' + name + '!'); // Hello, Kevin!

// With ES6 template literals.
console.log(`Hello, ${name}!`); // Hello, Kevin!
```

Within template literals you can evaluate expressions.

```
const price = 19.99;
const tax = 0.07;

const total = `The total price is ${price + (price * tax)}`;
console.log(total);
// The total price is 21.3893
```

Enhanced Object Literals

In ES5

```
const first = 'Kevin';  
const last = 'Langley';  
const age = 29;  
  
const person = {  
  first: first,  
  last: last,  
  age: age  
};
```

In ES6

```
const first = 'Kevin';  
const last = 'Langley';  
const age = 29;  
  
const person = {  
  first,  
  last,  
  age  
};
```

```
var obj = {  
  foo: function() {  
    console.log('foo');  
  },  
  bar: function() {  
    console.log('bar');  
  }  
};
```

```
const obj = {  
  foo() {  
    console.log('foo');  
  },  
  bar() {  
    console.log('bar');  
  }  
};
```

Typescript

Problems in JavaScript

- No compile time environment
- No data type for different type of values
- No OOP support
- Dynamic Typing
- Lack of modularity

Typescript

- Typescript is a language for application scale javascript development.
- It is a superset of JavaScript that compiles to plain JavaScript.
- Typescript = JavaScript + OOP + Data Types + Compile Time
- Optional Static Type Annotation / Static Typing
- Lightweight Productivity Booster
- Open Source
- Cross Platform

Getting Started

- `npm install -g typescript`

To compile typescript file:

- `tsc myfile.ts`

- Tsc -> Typescript compiler (creates js file)

Type Annotation

- Any :- It is a super set of all types
- Primitive :- number (int, float, double), String, Bool
- void
- Object Types :- class, interface, module
- Array
- Enum :- `enum color{red,green,blue};`
`var c=color.blue;`
- Tuple :- It allow you to express an array where the type of a fixed number of elements is known.
`var x:[number, boolean];x=[18,true];`
- Union:- `var isMarried: string | boolean;`

```
var isBatting: boolean = false;  
var inning: number = 6;  
var teamName: string = "Blue Jays";  
var runsRecord: number[] = [1, 6, 3];
```

```
var isBatting: any = false;  
var inning: any = 6;
```

```
var teamName: string = <any> false;  
var runsRecord: number[] = <any> false;
```

Function

- Type annotation for parameter and return type
- Optional and Default Parameter
- Function Overloads
- Fat Arrow functions
- Rest Parameters (...argumentname)

Type annotation for parameter and return type

```
var pitch = function(type: string): number {  
    if (type == "fastball")  
        return 170.5;  
    else  
        return 123.9;  
};
```

Optional Parameters:

```
function details(id:number, mail_id?:string) {  
    console.log("ID:", id);  
    console.log("Email Id",mail_id);  
}
```

Default Parameters:

```
function cal_discount(price:number,rate:number = 0.50) {  
    var discount = price * rate; console.log("Discount  
Amount: ",discount);  
}
```

Function Overloading

```
function add(a:string, b:string):string;
```

```
function add(a:number, b:number): number;
```

```
function add(a: any, b:any): any {  
    return a + b;  
}
```

```
add("Hello ", "Steve"); // returns "Hello Steve"
```

```
add(10, 20); // returns 30
```

Fat Arrow Function

Fat arrow notations are used for anonymous functions i.e for function expressions.

- Implicit return

Syntax: (param1, param2, ..., paramN) => expression

```
let sum = (x: number, y: number): number => {  
    return x + y;  
}
```

```
sum(10, 20); //returns 30
```

```
let Print = () => console.log("Hello TypeScript");
```

```
Print(); //Output: Hello TypeScript
```


Rest Parameter

When the number of parameters that a function will receive is not known or can vary, we can use rest parameters. In JavaScript, this is achieved with the "arguments" variable. However, with TypeScript, we can use the rest parameter denoted by ... (3 dots).

We can pass zero or more arguments to the rest parameter. The compiler will create an array of arguments with the rest parameter name provided by us.

```
function Greet(greeting: string, ...names: string[]) {  
    return greeting + " " + names.join(", ") + "!";  
}
```

```
Greet("Hello", "Steve", "Bill"); // returns "Hello Steve, Bill!"
```

```
Greet("Hello");// returns "Hello !"
```

TypeScript Classes

```
class Car {  
    //field  
    engine:string;  
  
    //constructor  
    constructor(engine:string) {  
        this.engine = engine  
    }  
  
    //function  
    disp():void {  
        console.log("Engine is : "+this.engine)  
    }  
}
```

```
//Generated by typescript 1.8.10  
var Car = (function () {  
    //constructor  
    function Car(engine) {  
        this.engine = engine;  
    }  
  
    //function  
    Car.prototype.disp = function () {  
        console.log("Engine is : " + this.engine);  
    };  
    return Car;  
})();
```

Constructor shortcut

```
class Employee{
    private name:string
    private basic:number
    private allowance:number

    constructor(name:string, basic:number, allowance:number){
        this.name = name
        this.basic = basic
        this.allowance = allowance
    }

    public getSalary():number{
        return this.basic + this.allowance
    }
}
```

```
1 class Employee{
2     constructor(public name:string, public basic: number,
3     public allowance:number){
4         // no initialization required
5     }
6
7     public getSalary(){
8         return this.basic + this.allowance;
9     }
10 }
11
12 var emp = new Employee('XYZ',100,10);
13 alert(emp.getSalary());
```

Inheritance

- Typescript supports inheritance of class through extends keyword

```
class Person{
    constructor(public name:string, public age:number){
    }
    showInfo(){
        alert("Name: " + this.name + " Age: " + this.age);
    }
} // Base class

class Employee extends Person{
    constructor(name, age, public salary:number){
        super(name,age); // super calls the constructor of base class
    }
    showInfo(){
        alert("Name:" + this.name + " Age:" + this.age + " Salary:" + this.salary)
    }
} // Class that inherits from Base class

var per:Person = new Person('Aniruddha',40);
per.showInfo(); // calls showInfo of Person class

var emp:Person = new Employee('Bill',55,100);
emp.showInfo(); // calls showInfo of Employee class
```

Interface

- Declared using interface keyword
- TS compiler shows error when interface signature and implementation does not match
- Optional properties can be declared for an interface (using ?)

```
Interface Employee{  
  Firstname:string;  
  LastName:string  
}
```
- Class implements interface.

END

Session-8

Topics to be covered...

- What is Node.js
- why Node.js
- What is NPM
- Node.js modules-Built-in and User defined
- Exports keyword
- Node.js Http module
- Node.js File System
- Url Module
- Node.js Events

What is Node.js

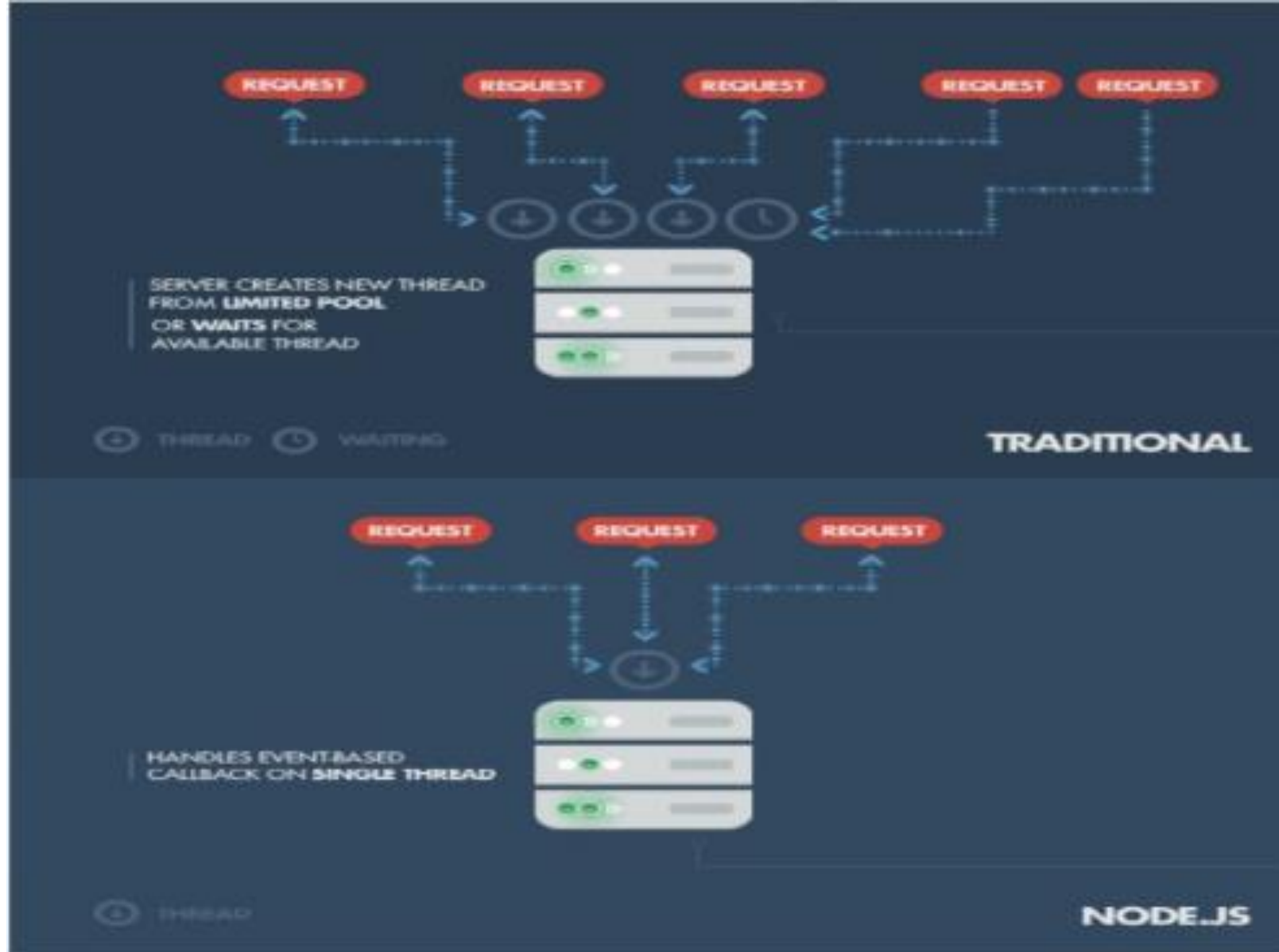
Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.

Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js = **Runtime** Environment + JavaScript Library

- Written in C++
- Built on top of Chrome's V8 engine –so pure JavaScript
- Framework to build asynchronous I/O applications
- Single Threaded – no concurrency bugs –no deadlock
- One node process = one CPU core

Blocking I/O vs. Non-Blocking I/O



Features of Node.js

Extremely fast: Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.

I/O is Asynchronous and Event Driven: All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.

Single threaded: Node.js follows a single threaded model with event looping.

Highly Scalable: Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.

No buffering: Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.

Global Object

`__dirname` :- It specifies the name of the directory that currently contains the code.

`__filename` : - It specifies the filename of the code being executed.

`setImmediate(callback[, arg][, ...])`

`setInterval(callback, delay[, arg][, ...])`

`setTimeout(callback, delay[, arg][, ...])`

`clearImmediate(immediateObject)`

`clearInterval(intervalObject)`

`clearTimeout(timeoutObject)`

Modules

A set of function you want to include in your application.

Consider modules to be the same as Javascript libraries.

Modules can either be single file or diectories containing one or more files

Types of Module

- Built In Module
- User Defined Custom Module (Using Exports Object)

How to create Module in Node.js

To create a typical module, you create a file that defines properties on the exports object with any kind of data, such as strings, objects and functions.

How to add Module in Node.js

Using require function built-in and custom modules may be added.

Ex

```
var http=require('http'); //Built-In Module
```

```
var dt=require('./custommodulename') //custom module
```

exports

Export Literals:

myModules.js

```
module.exports = 'Hello world';  
//or exports = 'Hello world';
```

app.js

```
var msg = require('./myModules.js');  
console.log(msg);
```

exports

Export Object :

myModules.js

```
exports.Message = 'Hello world';  
// or module.exports.Message = 'Hello world';  
  
exports.fn = function (msg) {  
    console.log(msg);  
};  
// or module.exports.fn = function (msg) { ... };
```

app.js

```
var msg = require('./myModules.js');  
msg.fn('Hello World');  
console.log(msg.Message);
```

HTTP Module

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

Syntax:

```
var http = require('http');
```

```
//create a server object:
```

```
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write('Hello'); //write a response to the client  
  res.end(); //end the response  
}).listen(1000); //the server object listens on port 1000
```

URL Module

The URL module splits up a web address into readable parts.

```
var url = require('url');  
var adr = 'http://localhost:1000/default.htm?id=2&name=ram';  
var q = url.parse(adr, true);  
  
console.log(q.host); //returns 'localhost:1000'  
console.log(q.pathname); //returns '/default.htm'  
console.log(q.search); //returns '?id=2&name=ram'  
  
var qdata = q.query; //returns an object: { id: 2, name: 'ram' }  
console.log(qdata.name); //returns 'ram'
```

File System

The Node.js file system module allow you to work with the file system on your computer.

Syntax:

```
var fs = require('fs');
```

```
fs.appendFile()
```

```
fs.readFile()
```

```
fs.readFileSync()
```

```
fs.writeFile()
```

```
fs.unlink(): delete a file with the File System module
```

```
fs.rename()
```

Events

Node.js allows us to create and handle custom events easily by using events module. Event module includes EventEmitter class which can be used to raise and handle custom events.

```
// get the reference of EventEmitter class of events module
var events = require('events');

//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();

//Subscribe for FirstEvent
em.on('FirstEvent', function (data) {
  console.log('First subscriber: ' + data); });

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter
example.');
```

END

Session-9

Topics to be covered...

- Node.js MySQL CRUD Operations
- Node.js MongoDB CRUD Operations

Mysql connect with node.js

```
npm install mysql --save
```

MySQL connection

app.js

```
var mysql = require('mysql');

var con = mysql.createConnection({           // Connection properties
  host: "192.168.100.26",
  user: "group000",
  port: "3306",
  password: "welcome",
  database: "group000"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.end();
});
```

create table

MySQL create table

```
var mysql = require('mysql');

var con = mysql.createConnection({           // Connection propertie
  host: "192.168.100.26",
  user: "group000",
  port: "3306",
  password: "welcome",
  database: "group000"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("create table a1 (c1 int, c2 varchar(10), c3 datetime)",
  function(err, result) {
    if (err) throw err;
    console.log("Table created!");
  })
  con.end();
});
```

insert record

MySQL insert record

```
var mysql = require('mysql');

var con = mysql.createConnection({          // Connection properties
  host: "192.168.100.26",
  user: "group000",
  port: "3306",
  password: "welcome",
  database: "group000"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("insert into dept values(1,1,1,1)", function(err, result){
    if (err) throw err;
    console.log("Record Inserted...");
  })
  con.end();
});
```

update record

MySQL update record

```
var mysql = require('mysql');

var con = mysql.createConnection({          // Connection properties
  host: "192.168.100.26",
  user: "group000",
  port: "3306",
  password: "welcome",
  database: "group000"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("update dept set dname = 'saleel' where deptno=1", function(err,
  result){
    if (err) throw err;
    console.log("Record Updated...");
  })
  con.end();
});
```

select

MySQL select record

```
var mysql = require('mysql');

var con = mysql.createConnection({           // Connection properties
  host: "192.168.100.26",
  user: "group000",
  port: "3306",
  password: "welcome",
  database: "group000"
});
con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM emp", function (err, data) {
    if (err) throw err;
    console.log(data);
  });
  con.end();
});
```

select

MySQL select record

```
var mysql = require('mysql');

var con = mysql.createConnection({          // Connection properties
  host: "192.168.100.26",
  user: "group000",
  port: "3306",
  password: "welcome",
  database: "group000"
});
con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM emp where job='manager'", function (err, data) {
    if (err) throw err;
    console.log(data);
  });
  con.end();
});
```


Mongodb connect with node.js

```
npm install mongodb --save
```

MongoDB create database:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/mydb";
MongoClient.connect(url, function(err, db)
{
  if (err) throw err;
  console.log("Database created!");
  db.close();
});
```

createCollection

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db)
{
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.createCollection("customers", function(err, res)
  {
    if (err) throw err;
    console.log("Collection created!");
    db.close();
  });
});
```

Insert

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db)
{
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = { name: "Company Inc", address: "Highway 37" };
  dbo.collection("customers").insertOne(myobj, function(err, res)
  {
    if (err) throw err;
    console.log("1 document inserted");
  });
  db.close();
});
```

find

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db)
{
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection("customers").findOne({}, function(err, result)
  {
    if (err) throw err;
    console.log(result.name);
    db.close();
  });
});
```

Find all

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, db)
{
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection("customers").find({}).toArray(function(err, result) {{
    if (err) throw err;
    console.log(result.name);
    db.close();
  }});
});
```

END

Session-10

ExpressJS

Express is a **minimal** and **flexible** node.js **web application framework**, providing a **robust** set of features for building **single** and **multi-page**, and **hybrid** web applications.

What is Express?

- Node.js based web framework
- Based on connect middleware
- Make usage of Node.js even easier
- Easy to implement REST API
- Easy to implement session management
- Supports several template rendering engines (jade, EJS, hbs)
 - support partials->so you can split your HTML in fragment
- Asynchronous
- Implements MVC pattern

- Allows to setup middleware's to responds to HTTP requests.
- Defines a routing table which is used t perform different action based on HTTP method and URL.
- Allows to dynamically render HTML pages based on passing arguments to templates.

```
var express = require('express');  
var app = express();  
app.get('/', function (req, res)  
{  
  res.send('Hello World!'); });  
app.listen(3000, function ()  
{  
  console.log('Example app listening on port 3000!');  
});
```

Advantages and Disadvantages

Advantages

- Regardless of complexity, there should be very few roadblocks if you know JavaScript well.
- Supports concurrency well

Disadvantages

- There is no built in error handling methods.

END

Thank you