**Introduction to JavaScript**

* JavaScript is a very powerful client-side scripting language. JavaScript is used mainly for enhancing the interaction of a user with the webpage. In other words, you can make your webpage more lively and interactive, with the help of JavaScript. JavaScript is also being used widely in game development and Mobile application development.
* JavaScript cannot run on its own. In fact, the browser is responsible for running JavaScript code. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it is up to the browser to execute it. They can be written right in a web page's HTML and run automatically as the page loads.
* Scripts are provided and executed as plain text. They don't need special preparation or compilation to run.

* JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.

The browser has an embedded engine sometimes called a "JavaScript virtual machine".

**How do engines work?**

Engines are complicated. But the basics are easy.

The engine (embedded if it's a browser) reads ("parses") the script.

Then it converts ("compiles") the script to the machine language.

And then the machine code runs, pretty fast.

The engine applies optimizations at each step of the process. It even watches the compiled script as it runs, analyzes the data that flows through it, and further optimizes the machine code based on that knowledge.

It does not provide low-level access to memory or CPU, because it was initially created for browsers which do not require it.

JavaScript's capabilities greatly depend on the environment it's running in. For instance, Node.js supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc.

**Difference Between JavaScript vs VBScript**

* JavaScript is not a true object-oriented scripting language as it doesn't support inheritance concept, subroutines but it supports usage of an object, definition of classes for subsequent object creation whereas VBScript also not a true object-oriented scripting language as it doesn't support inheritance, object usage, classes usage but it supports reusable functions and subroutines.
* JavaScript is a default scripting language for most of the browsers whereas VBScript is not a default scripting language and we need to mention it as a scripting language.
* JavaScript is case-sensitive scripting language whereas VBScript is not a case-sensitive scripting language.
* JavaScript syntax is similar to the C programming language whereas VBScript syntax is similar to the Visual Basic as it is a subpart of it and it follows the syntax of the visual basic.
* JavaScript is used as a client-side scripting language whereas VBScript can be used as both server-side and client-side scripting language.
* JavaScript uses the same operator for different operations whereas VBScript uses different operators for different operations.

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript . local variable and global variable.

**JavaScript Statement**

**The following table lists all JavaScript statements.**

* **break** Exits a switch or a loop

* **const** Declares a variable with a constant value

* **class** Declares a class

* **continue** Breaks one iteration (in the loop) if a specified condition occurs, and continues with the next iteration in the loop

* **debugger** Stops the execution of JavaScript, and calls (if available) the debugging function

* **do ... while** Executes a block of statements and repeats the block while a condition is true

* **for** Loops through a block of code a number of times

* **for ... in** Loops through the properties of an object

* **for ... of** Loops through the values of an iterable object

* **function** Declares a function

* **if ... else ... else if** Marks a block of statements to be executed depending on a condition

* **let** Declares a variable inside brackets {} scope

* **return** Stops the execution of a function and returns a value from that function

* **switch** Marks a block of statements to be executed depending on different cases

* **throw** Throws (generates) an error

* **try ... catch ... finally** Marks the block of statements to be executed when an error occurs in a try block, and implements error handling

* **var** Declares a variable

* **while** Marks a block of statements to be executed while a condition is true

**Arithmetic Operators**

+ (Addition) - (Subtraction) * (Multiplication) / (Division)

% (Modulus) ++ (Increment) -- (Decrement)

**Comparison Operators**

= = (Equal) != (Not Equal) > (Greater than) < (Less than)

<= (Less than or Equal to) >= (Greater than or Equal to)

**Logical Operators**

&& (Logical AND) || (Logical OR) ! (Logical NOT)

**Bitwise Operators**

& (Bitwise AND) | (BitWise OR) ^ (Bitwise XOR) ~ (Bitwise Not)

<< (Left Shift) >> (Right Shift) >>> (Right shift with Zero)

**Assignment Operators**

= (Simple Assignment ) += (Add and Assignment) –= (Subtract and Assignment)

*= (Multiply and Assignment) /= (Divide and Assignment)

%= (Modules and Assignment)

**Miscellaneous Operator66**

? . (Conditional )

**typeof Operator .** The typeof operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand. The typeof operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

**Comments**

Single Line //

Multi-line /* */

**Construct.**

Construct is a generic term referring to an arbitrary aggregate of code in a specific formation. It is not a javascript-specific term. Basically, it can apply to anything. So, while the code you referenced is a construct known as a self invoking anonymous function, var x = "hello world"; is a construct known as a variable declaration and assignment.

Constructs are the structures that you can use in a JavaScript to control the flow of the script

JavaScript Function .

```
function myFunction(a, b) {
  return a * b;           // Function returns the product of a and b     }
```

***** Variables declared within a JavaScript function, become LOCAL to the function.

**JavaScript Display Possibilities**

**JavaScript can "display" data in different ways.**

1) Writing into an HTML element, using **innerHTML.**

To access an HTML element, JavaScript can use the document.getElementById(id) method.

2) Writing into the HTML output using **document.write().**

Using document.write() after an HTML document is loaded, will delete all existing HTML.

3) Writing into an alert box, using **window.alert().**

You can use an alert box to display data.

4) Writing into the browser console, using **console.log().**

For debugging purposes, you can use the console.log() method to display data.

**JavaScript Function Scope**

Local scope Global scope

* Local variables have Function scope

* A variable declared outside a function, becomes GLOBAL. A global variable has global scope. All scripts and functions on a web page can access it. If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable.

* In JavaScript, objects and functions are also variables.

**Context**

Many developers often confuse scope and context as if they equally refer to the same concepts. But this is not the case. Scope is what we discussed above and Context is used to refer to the value of this in some particular part of your code. Scope refers to the visibility of variables and context refers to the value of this in the same scope. We can also change the context using function methods, which we will discuss later. In the global scope context is always the Window object.

---

* In "**Strict Mode**", undeclared variables are not automatically global. **"use strict";** Defines that JavaScript code should be executed in "strict mode". It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.The purpose of "use strict" is to indicate that the code should be executed in "strict mode". With strict mode, you can not, for example, use undeclared variables.

* Strict mode makes it easier to write "secure" JavaScript. Strict mode changes previously accepted "bad syntax" into real errors. As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable. In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties. In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.

**JavaScript String Methods**

* The length property returns the length of a string.

* The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string.JavaScript counts positions from zero.

* The lastIndexOf() method returns the index of the last occurrence of a specified text in a string.

Both indexOf(), and lastIndexOf() return -1 if the text is not found.

Both methods accept a second parameter as the starting position for the search.

* The search() method searches a string for a specified value and returns the position of the match.

**+ The two methods, indexOf() and search(), are equal?**

They accept the same arguments (parameters), and return the same value?

The two methods are NOT equal. These are the differences.

The search() method cannot take a second start position argument.

The indexOf() method cannot take powerful search values (regular expressions).

+ There are 3 methods for extracting a part of a string.

**1-- slice(start, end)** . slice() extracts a part of a string and returns the extracted part in a new string. The method takes 2 parameters. the start position, and the end position (end not included).If a parameter is negative, the position is counted from the end of the string.

If you omit the second parameter, the method will slice out the rest of the string.

**substring(start, end)**----substring() is similar to slice().The difference is that substring() cannot accept negative indexes.If you omit the second parameter, substring() will slice out the rest of the string.

**substr(start, length)**----substr() is similar to slice().The difference is that the second parameter specifies the length of the extracted part.If you omit the second parameter, substr() will slice out the rest of the string.If the first parameter is negative, the position counts from the end of the string.

* The **replace()** method replaces a specified value with another value in a string.The replace() method does not change the string it is called on. It returns a new string.By default, the replace() method replaces only the first match.By default, the replace() method is case sensitive.

Note . To replace case insensitive, use a regular expression with an /i flag (insensitive). Note that regular expressions are written without quotes.To replace all matches, use a regular expression with a / g flag (global match)

*A string is converted to upper case with **toUpperCase().**

*A string is converted to lower case with **toLowerCase().**

***concat()** joins two or more strings.The concat() method can be used instead of the plus operator. These two lines do the same.

**All string methods return a new string. They don't modify the original string.Formally said. Strings are immutable. Strings cannot be changed, only replaced.**

* The **trim()** method removes whitespace from both sides of a string.

*There are 3 methods for extracting string characters.

1---**charAt(position)** ---The charAt() method returns the character at a specified index (position) in a string.

**2----charCodeAt(position)**----The charCodeAt() method returns the unicode of the character at a specified index in a string.The method returns a UTF-16 code (an integer between 0 and 65535).

*A string can be converted to an array with the split() method.

**NUMBERS**

*JavaScript has only one type of number. Numbers can be written with or without decimals.Extra large or extra small numbers can be written with scientific (exponent) notation.JavaScript Numbers are Always 64-bit Floating Point.JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.JavaScript numbers are always stored as double precision floating point numbers,Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

*If you add a number and a string, the result will be a string concatenation.

*JavaScript will try to convert strings to numbers in all numeric operations.

eg var x = "100"; var y = "10";

var z = x / y; // z will be 10 **OKKKKKK**

eg But this will not work.

var x = "100"; var y = "10";

var z = x + y; // z will not be 110 (It will be 10010)

***NaN** is a JavaScript reserved word indicating that a number is not a legal number.Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number). However, if the string contains a numeric value , the result will be a number..You can use the global JavaScript function isNaN() to find out if a value is a number.Watch out for NaN. If you use NaN in a mathematical operation, the result will also be NaN.

NaN is a number. typeof NaN returns number

*** Infinity (or -Infinity)** is the value JavaScript will return if you calculate a number outside the largest possible number.Division by 0 (zero) also generates Infinity. Infinity is a number. typeof Infinity returns number.

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

eg . var x = 0xFF;  // x will be 255

* By default, JavaScript displays numbers as base 10 decimals.But you can use the toString() method to output numbers from base 2 to base 36.

**+** Hexadecimal is base 16. Decimal is base 10. Octal is base 8. Binary is base 2.

*Normally JavaScript numbers are primitive values created from literals.

var x = 123;

But numbers can also be defined as objects with the keyword new.

var y = new Number(123);

* When using the == operator, equal numbers are equal.

*When using the === operator, equal numbers are not equal, because the === operator expects equality in both type and value.

**NUMBER METHODS**

All number methods can be used on any type of numbers (literals, variables, or expressions).

* The **toString()** method returns a number as a string.

***toExponential()** returns a string, with a number rounded and written using exponential notation.A parameter defines the number of characters behind the decimal point.

* **toFixed()** returns a string, with the number written with a specified number of decimals.

***toPrecision()** returns a string, with a number written with a specified length.

***valueOf()** returns a number as a number.In JavaScript, a number can be a primitive value (typeof = number) or an object (typeof = object).The valueOf() method is used internally in JavaScript to convert Number objects to primitive values.

*There are **3** JavaScript methods that can be used to convert variables to numbers.

These methods are not number methods, but global JavaScript methods

1--The **Number()** method---Number() can be used to convert JavaScript variables to numbers.

If the number cannot be converted, NaN (Not a Number) is returned.Number() can also convert a date to a number.

2--The **parseInt()** method---parseInt() parses a string and returns a whole number. Spaces are allowed. Only the first number is returned.If the number cannot be converted, NaN (Not a Number) is returned.

3--The **parseFloat()** method----parseFloat() parses a string and returns a number. Spaces are allowed. Only the first number is returned.If the number cannot be converted, NaN (Not a Number) is returned.

*MAX_VALUE returns the largest possible number in JavaScript.

*MIN_VALUE returns the lowest possible number in JavaScript.

*POSITIVE_INFINITY is returned on overflow.

*NEGATIVE_INFINITY is returned on underflow.

+ Number properties belongs to the JavaScript's number object wrapper called number. These properties can only be accessed as Number.MAX_VALUE.

**Date**

* Date objects are created with the new Date() constructor.new Date() creates a new date object with the current date and time.Date objects are static. The computer time is ticking, but date objects are not.

There are **4** ways to create a new date object.

* new Date()  * new Date(milliseconds)  * new Date(date string)

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

*The **getTime()** method returns the number of milliseconds since January 1, 1970.

*The **getFullYear()** method returns the year of a date as a four digit number.

*The **getMonth()** method returns the month of a date as a number (0-11).You can use an array of names, and getMonth() to return the month as a name.

In JavaScript, the first month (January) is month number 0, so December returns month number 11.

*The **getDate()** method returns the day of a date as a number (1-31).

*The **getHours()** method returns the hours of a date as a number (0-23).

*The **getMinutes()** method returns the minutes of a date as a number (0-59).

*The **getSeconds()** method returns the seconds of a date as a number (0-59).

*The **getMilliseconds()** method returns the milliseconds of a date as a number (0-999).

*The **getDay()** method returns the weekday of a date as a number (0-6).You can use an array of names, and getDay() to return the weekday as a name.

**JavaScript Arrays**

**eg  1--**var array_name = [item1, item2, ...];  2--var cars = new Array("Saab", "Volvo", "BMW");

****the full array can be accessed by referring to the array name.

```
var cars = ["Saab", "Volvo", "BMW"];

document.getElementById("demo").innerHTML = cars;
```

*Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.

*JavaScript variables can be objects. Arrays are special kinds of objects.

```
eg  myArray[0] = Date.now; myArray[1] = myFunction;  myArray[2] = myCars;
```

*The safest way to loop through an array, is using a for loop.

```
eg .  for (i = 0; i < fLen; i++) {

 text += "<li>" + fruits[i] + "</li>";  }
```

*You can also use the Array.forEach() function.

```
eg fruits.forEach(myFunction);
```

*The easiest way to add a new element to an array is using the push() method.

--New element can also be added to an array using the length property.

*In JavaScript, arrays use numbered indexes.

***I**n JavaScript, objects use named indexes.

*  JavaScript does not support associative arrays. You should use objects when you want the element names to be strings (text).  You should use arrays when you want the element names to be numbers.

*The **join()** method also joins all array elements into a string.It behaves just like toString(), but in addition you can specify the separator.

*The **pop()** method removes the last element from an array. The pop() method returns the value that was "popped out".

*The **push()** method adds a new element to an array (at the end).The push() method returns the new array length.

*The **shift()** method removes the first array element and "shifts" all other elements to a lower index.Shifting is equivalent to popping, working on the first element instead of the last.The shift() method returns the string that was "shifted out".

*The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements. The unshift() method returns the new array length.

*Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete.Using delete may leave undefined holes in the array. Use pop() or shift() instead.

*The **splice()** method can be used to add new items to an array. you can use splice() to remove elements without leaving "holes" in the array.

*The **slice()** method slices out a piece of an array into a new array.The slice() method creates a new array. It does not remove any elements from the source array.The slice() method can take two arguments like slice(1, 3).The method then selects elements from the start argument, and up to (but not including) the end argument.If the end argument is omitted, the slice() method slices out the rest of the array.

**Boolean Values**

JavaScript has a Boolean data type. It can only take the values true or false.

*You can use the Boolean() function to find out if an expression (or a variable) is true.

*Everything Without a "Value" is False

eg -The Boolean value of 0 (zero) is false.  -The Boolean value of null is false.

-The Boolean value of -0 (minus zero) is false.-The Boolean value of NaN is false.

-The Boolean value of "" (empty string) is false.

-The Boolean value of undefined is false.

***Comparisons**

For following consider x= 5;

| | | | |
|---|---|---|---|
| **===** | equal value and equal type . | x === 5 | true |
| | | x === "5" | false |
| !== | not equal value or not equal type | x !== 5 | false |
| | | x !== "5" | true |
| | | x !== 8 | true |

***Conditional Statements ----if else , for etc.....Same like c++ , java etc**

***Function Invocation**

The code inside the function will execute when "something" invokes (calls) the function.

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

**javascript Function Closures**

The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope.This is called a JavaScript closure. It makes it possible for a function to have "private" variables.The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

**JavaScript Objects**

objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of “key. value” pairs.In JavaScript, almost "everything" is an object. see following eg

* Booleans can be objects (if defined with the new keyword)
* Dates are always objects
* Numbers can be objects (if defined with the new keyword)
* Maths are always objects
*Strings can be objects (if defined with the new keyword)
* Arrays are always objects
* Regular expressions are always objects
* Functions are always objects
* Objects are always objects

**+++**A primitive value is a value that has no properties or methods.A primitive data type is data that has a primitive value.

**Types.** string number boolean null undefined

*Objects are mutable. They are addressed by reference, not by value.

// Adding or changing an object property

Object.defineProperty(object, property, descriptor)

// Adding or changing many object properties

Object.defineProperties(object, descriptors)

// Accessing Properties

Object.getOwnPropertyDescriptor(object, property)

// Returns all properties as an array

Object.getOwnPropertyNames(object)

// Returns enumerable properties as an array

Object.keys(object)

// Accessing the prototype

Object.getPrototypeOf(object)

// Prevents adding properties to an object

Object.preventExtensions(object)

// Returns true if properties can be added to an object

Object.isExtensible(object)

// Prevents changes of object properties (not values)

Object.seal(object)

// Returns true if object is sealed

Object.isSealed(object)

// Prevents any changes to an object

Object.freeze(object)

// Returns true if object is frozen

Object.isFrozen(object)

****Object Prototypes---**

****NOTE*****In JavaScript, inheritance is supported by using prototype object. Some people call it "Prototypal Inheriatance" and some people call it "Behaviour Delegation".

JavaScript prototypes are used to accessing properties and methods of objects. Inherited properties are originally defined in the prototype or parent object. The Date object is inherited from Date.prototype, Array object inherits from Array.prototype etc. The prototypes may be used to add new properties and methods to the existing objects and object constructor.

Syntax. Object.prototype

The Object.prototype is on the top of the prototype inheritance chain.The JavaScript prototype property also allows you to add new methods to objects constructors.All JavaScript objects inherit properties and methods from a prototype. eg

* Date objects inherit from Date.prototype * Array objects inherit from Array.prototype

**JavaScript Object Constructors**

**EX----**function Person(first, last) { this.firstName = first; this.lastName = last; }

*It is considered good practice to name constructor functions with an upper-case first letter.

*constructor function can also define methods.cannot add a new method to an object constructor the same way you add a new method to an existing object.Adding methods to an object constructor must be done inside the constructor function.

**JavaScript Encapsulation**

The JavaScript Encapsulation is a process of binding the data (i.e. variables) with the functions acting on that data. It allows us to control the data and validate it. To achieve an encapsulation in JavaScript. -

Use var keyword to make data members private.

Use setter methods to set the data and getter methods to get that data.

*The encapsulation allows us to handle an object using the following properties.

**Read/Write** - Here, we use setter methods to write the data and getter methods read that data.

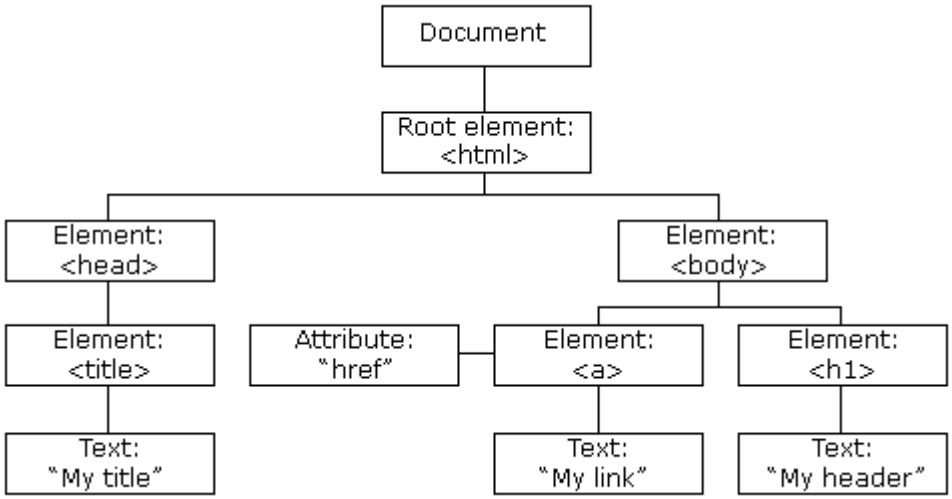
**JavaScript Abstraction**

An abstraction is a way of hiding the implementation details and showing only the functionality to the users. In other words, it ignores the irrelevant details and shows only the required one.

* Points to remember

We cannot create an instance of Abstract Class. It reduces the duplication of code

**The HTML DOM (Document Object Model)**




With the object model, JavaScript gets all the power it needs to create dynamic HTML.

JavaScript can change all the HTML elements in the page

JavaScript can change all the HTML attributes in the page

JavaScript can change all the CSS styles in the page

JavaScript can remove existing HTML elements and attributes

JavaScript can add new HTML elements and attributes

JavaScript can react to all existing HTML events in the page

JavaScript can create new HTML events in the page

*"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

**The W3C DOM standard is separated into 3 different parts.**

**Core DOM** - standard model for all document types

**XML DOM** - standard model for XML documents

**HTML DOM** - standard model for HTML documents

The HTML DOM is a standard object model and programming interface for HTML. It defines.

1) The HTML elements as objects
2) The properties of all HTML elements
3) The methods to access all HTML elements
4) The events for all HTML elements

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

HTML DOM methods are actions you can perform (on HTML Elements).HTML DOM properties are values (of HTML Elements) that you can set or change.In the DOM, all HTML elements are defined as objects.A property is a value that you can get or set (like changing the content of an HTML element).A method is an action you can do (like add or deleting an HTML element).

**The getElementById Method**

The most common way to access an HTML element is to use the id of the element.In the example above the getElementById method used id="demo" to find the element.

**The innerHTML Property**

The easiest way to get the content of an element is by using the innerHTML property.The innerHTML property is useful for getting or replacing the content of HTML elements.The innerHTML property can be used to get or change any HTML element, including <html> and <body>.

**The HTML DOM Document Object**

The document object represents your web page.If you want to access any element in an HTML page, you always start with accessing the document object.Below are some examples of how you can use the document object to access and manipulate HTML.

**Finding HTML Elements**

+ document.getElementById(id)  Find an element by element id

+ document.getElementsByTagName(name)  Find elements by tag name

+ document.getElementsByClassName(name)  Find elements by class name

+ document.getElementsByTagName("p");  Finding HTML Elements by Tag Name

+ document.querySelectorAll("p.intro");  If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

**Changing HTML Elements**

+ element.innerHTML =  new html content  Change the inner HTML of an element

+ element.attribute = new value  Change the attribute value of an HTML element

+ element.style.property = new style  Change the style of an HTML element

+ element.setAttribute(attribute, value)  Change the attribute value of an HTML element

**Adding and Deleting Elements**

+ document.createElement(element)  Create an HTML element

+ document.removeChild(element)  Remove an HTML element

+ document.appendChild(element)  Add an HTML element

+ document.replaceChild(new, old)  Replace an HTML element

+ document.write(text)  Write into the HTML output stream

### To change the style of an HTML element, use this syntax.

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

### Add an event listener that fires when a user clicks a button.

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

The **addEventListener()** method attaches an event handler to the specified element.The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.You can add event listeners to any DOM object not only HTML elements. i.e the window object.The addEventListener() method makes it easier to control how the event reacts to bubbling.When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.You can easily remove an event listener by using the removeEventListener() method.

*The **addEventListener()** method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object, or other objects that support events, like the xmlHttpRequest object.

**EX--**Add an event listener that fires when a user resizes the window.

```
window.addEventListener("resize", function(){

 document.getElementById("demo").innerHTML = sometext;});
```

**The **removeEventListener()** method removes event handlers that have been attached with the addEventListener() method

**##The HTMLCollection Object**

The getElementsByTagName() method returns an HTMLCollection object.An HTMLCollection object is an array-like list (collection) of HTML elements.

********** READ DOM REALTED ON NET FOR MORE DETAILS *********

**checkValidity()** Returns true if an input element contains valid data.

**setCustomValidity()** Sets the validationMessage property of an input element.

**Browser Object Model**

* The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.The object of window represents a browser window and all its corresponding features. A window object is created automatically by the browser itself.Java Script’s window.screen object contains information about the user’s screen

***Window Object**

The window object is supported by all browsers. It represents the browser's window.

All global JavaScript objects, functions, and variables automatically become members of the window object.Global variables are properties of the window object.Global functions are methods of the window object.Even the document object (of the HTML DOM) is a property of the window object.

```
window.document.getElementById("header");

window.innerHeight - the inner height of the browser window (in pixels)

window.innerWidth - the inner width of the browser window (in pixels)

window.moveTo() - move the current window

window.resizeTo() - resize the current window
```

**Cookie with JavaScript**

* JavaScript can create, read, and delete cookies with the document.cookie property.

You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed.

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12.00.00 UTC";
```

*document.cookie will return all cookies in one string much like. cookie1=value; cookie2=value; cookie3=value;

**Timing Events**

*setTimeout(function, milliseconds)  ---- Executes a function, after waiting a specified number of milliseconds.The window.setTimeout() method can be written without the window prefix.

*setInterval(function, milliseconds)----  Same as setTimeout(), but repeats the execution of the function continuously.

The setTimeout() and setInterval() are both methods of the HTML DOM Window object.

*The clearTimeout() method stops the execution of the function specified in setTimeout().

**Alert Box** **eg** window.alert("sometext");

The window.alert() method can be written without the window prefix.

*A confirm box is often used if you want the user to verify or accept something.

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

*A prompt box is often used if you want the user to input a value before entering a page.

```
window.prompt("sometext","defaultText");
```

**Window Navigator**

The window.navigator object contains information about the visitor's browser.

+The cookieEnabled property returns true if cookies are enabled, otherwise false.

*history.back() - same as clicking back in the browser.The history.back() method loads the previous URL in the history list.

* history.forward() - same as clicking forward in the browser.The history.forward() method loads the next URL in the history list.

+ JSLint. According to its website, JSLint is a “JavaScript Code Quality Tool.”

---

**jQuery**

jQuery is a lightweight, "write less, do more", JavaScript library.jQuery is an open source JavaScript library that simplifies the interactions between an HTML/CSS document, or more precisely the Document Object Model (DOM), and JavaScript.Elaborating the terms, jQuery simplifies HTML document traversing and manipulation, browser event handling, DOM animations, Ajax interactions, and cross-browser JavaScript development.

**Why jQuery?**

* It is incredibly popular, which is to say it has a large community of users and a healthy amount of contributors who participate as developers and evangelists.

* It normalizes the differences between web browsers so that you don’t have to.

* It is intentionally a lightweight footprint with a simple yet clever plugin architecture.

* Its repository of plugins is vast and has seen steady growth since jQuery's release.

* Its API is fully documented, including inline code examples, which in the world of JavaScript libraries is a luxury. Heck, any documentation at all was a luxury for years.

* It is friendly, which is to say it provides helpful ways to avoid conflicts with other JavaScript libraries.

**Basic syntax for any jQuery function is.**

```
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();          });     });
```

**Advantages.**

+ Wide range of plug-ins. jQuery allows developers to create plug-ins on top of the JavaScript library.

+ Large development community

+ It has a good and comprehensive documentation

+ It is a lot more easy to use compared to standard javascript and other javascript libraries.

+ JQuery lets users develop Ajax templates with ease, Ajax enables a sleeker interface where actions can be performed on pages without requiring the entire page to be reloaded.

+ Being Light weight and a powerful chaining capabilities makes jQuery more strong.

**Disadvantages.**

+ While JQuery has an impressive library in terms of quantity, depending on how much customization you require on your website, functionality maybe limited thus using raw javascript maybe inevitable in some cases.

+ The JQuery javascript file is required to run JQuery commands, while the size of this file is relatively small (25-100KB depending on server), it is still a strain on the client computer and maybe your web server as well if you intend to host the JQuery script on your own web server.

**Selector :** The jQuery #id selector uses the id attribute of an HTML tag to find the specific element.

```
$("#test")
```

**jQuery select() Method**

The select event occurs when a text is selected (marked) in a text area or a text field.

The select() method triggers the select event, or attaches a function to run when a select event occurs.

```
$("input").select(function(){
alert("Text marked!");                    });
```

*The jQuery animate() method is used to create custom animations.

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

***Traversing**

jQuery traversing, which means "move through", are used to "find" (or select) HTML elements based on their relation to other elements. Start with one selection and move through that selection until you reach the elements you desire.

The image below illustrates an HTML page as a tree (DOM tree). With jQuery traversing, you can easily move up (ancestors), down (descendants) and sideways (siblings) in the tree, starting from the selected (current) element. This movement is called traversing - or moving through - the DOM tree.

**jQuery - DOM Manipulation**

JQuery provides methods to manipulate DOM in efficient way. You do not need to write big code to modify the value of any element's attribute or to extract HTML code from a paragraph or division.

JQuery provides methods such as .attr(), .html(), and .val() which act as getters, retrieving information from DOM elements for later use.

**Content Manipulation**

The html( ) method gets the html contents (innerHTML) of the first matched element.

Here is the syntax for the method – selector.html( )

**DOM Element Replacement**

You can replace a complete DOM element with the specified HTML or DOM elements. The replaceWith( content ) method serves this purpose very well.

Here is the syntax for the method – selector.replaceWith( content )

Removing DOM Elements

There may be a situation when you would like to remove one or more DOM elements from the document. JQuery provides two methods to handle the situation.

The empty( ) method remove all child nodes from the set of matched elements where as the method remove( expr ) method removes all matched elements from the DOM.

Here is the syntax for the method −selector.remove( [ expr ])

or selector.empty( )

**Inserting DOM Elements**

There may be a situation when you would like to insert new one or more DOM elements in your existing document. JQuery provides various methods to insert elements at various locations.

The after( content ) method insert content after each of the matched elements where as the method before( content ) method inserts content before each of the matched elements.

Here is the syntax for the method −selector.after( content )

or selector.before( content )

****A plug-in is piece of code written in a standard JavaScript file. These files provide useful jQuery methods which can be used along with jQuery library methods.

To make a plug-in's methods available to us, we include plug-in file very similar to jQuery library file in the <head> of the document.We must ensure that it appears after the main jQuery source file, and before our custom JavaScript code.

**Google Web Toolkit (GWT) is a development toolkit for building and optimizing complex browser-based applications. GWT is used by many products at Google, including Google AdWords and Orkut. GWT is an open source, completely free, and used by thousands of developers around the world

**Node.js Introduction**

Node.js is an open source server environment. Node.js is free. Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.). Node.js uses JavaScript on the server.Node.js is an open source and cross-platform runtime environment for executing JavaScript code outside of a browser. You need to remember that NodeJS is not a framework and it's not a programming language. Most of the people are confused and understand it's a framework or a programming language

**Features of NodeJS.**

- It's easy to get started and can be used for prototyping and agile development
- It provide fast and highly scalable services
- It uses JavaScript everywhere so it's easy for a JavaScript programmer to build back-end services using Node.js
- Source code more cleaner and consistent.
- Large ecosystem for open source library.
- It has Asynchronous or Non blocking nature.

**Advantages of NodeJS.**

1.Easy Scalability. 2.Real time web apps 3.Fast Suite

4.Easy to learn and code 4.Advantage of Caching 6.Data Streaming

7.Hosting 8.Corporate Support

**NodeJS should be preferred to build.**

Real Time Chats, Complex Single-Page applications, Real-time collaboration tools,

Streaming apps JSON APIs based application

**Node.js Modules**

modules in Node js are a way of encapsulating code in a separate logical unit. There are many readymade modules available in the market which can be used within Node js.NPM (Node Package Manager) is the default package manager for Node.js and is written entirely in Javascrip

*To include a module, use the require() function with the name of the module.

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

*npm is open source

**What Can Node.js Do?**

Node.js can generate dynamic page content

Node.js can create, open, read, write, delete, and close files on the server

Node.js can collect form data

Node.js can add, delete, modify data in your database

**What is a Node.js File?**

Node.js files contain tasks that will be executed on certain events

A typical event is someone trying to access a port on the server

Node.js files must be initiated on the server before having any effect

Node.js files have extension ".js"

---

**Introduction to Express**

Express is a small framework that sits on top of Node.js's web server functionality to simplify its APIs and add helpful new features.It makes it easier to organize your application's functionality with middle ware and routing; it adds helpful utilities to Node.js's HTTP objects;it facilitates the rendering of dynamic HTTP objects.

Express is a part of MEAN stack, a full stack JavaScript solution used in building fast, robust, and maintainable production web applications.

Express is a routing and middleware web framework that has minimal functionality of its own. An Express application is essentially a series of middleware function calls.

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
res.send('Hello World');
})
var server = app.listen(8081, function () {
var host = server.address().address
var port = server.address().port
  console.log("Example app listening at http.//%s.%s", host, port)
})
```

**TO RUN -$ node server.js**
**$ npm install express --save**

Express application uses a callback function whose parameters are **request** and **response** objects.

```
app.get('/', function (req, res)
{       // --                      })
```

- Request Object – The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.
- Response Object – The response object represents the HTTP response that an Express app sends when it gets an HTTP request.

*Middleware* functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle. The next middleware function is commonly denoted by a variable named next.

Middleware functions can perform the following tasks.

Execute any code.

- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware function in the stack.

If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

An Express application can use the following types of middleware.

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

The app object conventionally denotes the Express application. Create it by calling the top-level express() function exported by the Express module.

The app object has methods for

- Routing HTTP requests; see for example, app.METHOD and app.param.
- Configuring middleware; see app.route.
- Rendering HTML views; see app.render.
- Registering a template engine; see app.engine.

The app.locals object has properties that are local variables within the application. Once set, the value of app.locals properties persist throughout the life of the application, in contrast with res.locals properties that are valid only for the lifetime of the request.

**app.use([path,] callback [, callback...])**

Mounts the specified middleware function or functions at the specified path. the middleware function is executed when the base of the requested path matches path.

```
app.use(function (req, res, next) {
  console.log('Time. %d', Date.now())
  next()
})
```

| | |
|---|---|
| | Callback functions; can be.<br><br>• A middleware function.<br>• A series of middleware functions (separated by commas).<br>• An array of middleware functions.<br>• A combination of all of the above. |

**Request**

The req object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on. In this documentation and by convention, the object is always referred to as req (and the HTTP response is res) but its actual name is determined by the parameters to the callback function in which you're working.

For example.  app.get('/user/.id', function (req, res) { res.send('user ' + req.params.id) })

The req object is an enhanced version of Node's own request object and supports all built-in fields and methods.

**req.app**

This property holds a reference to the instance of the Express application that is using the middleware.If you follow the pattern in which you create a module that just exports a middleware function and require() it in your main file, then the middleware can access the Express instance via req.app

**Response**

The res object represents the HTTP response that an Express app sends when it gets an HTTP request.In this documentation and by convention, the object is always referred to as res (and the HTTP request is req) but its actual name is determined by the parameters to the callback function in which you're working.

For example.

```
app.get('/user/.id', function (req, res) {
  res.send('user ' + req.params.id)
})
```

**res.app**

This property holds a reference to the instance of the Express application that is using the middleware.

res.app is identical to the req.app property in the request object.

** Use the npm init command to create a package.json file for your application. This command prompts you for a number of things, including the name and version of your application and the name of the initial entry point file (by default this is **index.js**)

- Now install Express in the myapp directory and save it in the dependencies list of your **package.json** file
- npm install express

You can start the server by calling node with the script in your command prompt.

```
>node index.js
```

The Express Application Generator tool generates an Express application "skeleton". Install the generator using NPM

```
npm install express-generator -g
```

To create an *Express* app named "helloworld" with the default settings, navigate to where you want to create it and run the app as shown.

```
express helloworld
```

**sendFile() function.**

ExpressJS provides **sendFile()** function which will basically send HTML files to browser which then automatically interpreted by browser. All we need to do is in every route deliver appropriate HTML file.

**AngularJS**

AngularJS is a structural framework for dynamic web apps.  AngularJS is an open source Model-View-Controller framework which is similar to the JavaScript framework. This framework is used for

developing mostly Single Page applications .AngularJS is a JavaScript framework written in JavaScript.AngularJS is distributed as a JavaScript file, and can be added to a web page with a script tag.

It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.

AngularJS is what HTML would have been, had it been designed for applications. HTML is a great declarative language for static documents. It does not contain much in the way of creating applications, and as a result building web applications is an exercise in *what do I have to do to trick the browser into doing what I want?*

The impedance mismatch between dynamic applications and static documents is often solved with.

- **a library** - a collection of functions which are useful when writing web apps. Your code is in charge and it calls into the library when it sees fit. E.g., jQuery.
- **frameworks** - a particular implementation of a web application, where your code fills in the details. The framework is in charge and it calls into your code when it needs something app specific. E.g., durandal, ember, etc.

AngularJS takes another approach. It attempts to minimize the impedance mismatch between document centric HTML and what an application needs by creating new HTML constructs. AngularJS teaches the browser new syntax through a construct we call *directives*. Examples include.

- Data binding, as in {{}}.
- DOM control structures for repeating, showing and hiding DOM fragments.
- Support for forms and form validation.
- Attaching new behavior to DOM elements, such as DOM event handling.
- Grouping of HTML into reusable components.

**A complete client-side solution**

AngularJS is not a single piece in the overall puzzle of building the client-side of a web application. It handles all of the DOM and AJAX glue code you once wrote by hand and puts it in a well-defined structure. This makes AngularJS opinionated about how a CRUD (Create, Read, Update, Delete) application should be built. But while it is opinionated, it also tries to make sure that its opinion is just a starting point you can easily change. AngularJS comes with the following out-of-the-box.

- Everything you need to build a CRUD app in a cohesive set. Data-binding, basic templating directives, form validation, routing, deep-linking, reusable components and dependency injection.
- Testability story. Unit-testing, end-to-end testing, mocks and test harnesses.
- Seed application with directory layout and test scripts as a starting point.

**AngularJS's sweet spot**

AngularJS simplifies application development by presenting a higher level of abstraction to the developer. Like any abstraction, it comes at a cost of flexibility. In other words, not every app is a good fit for AngularJS. AngularJS was built with the CRUD application in mind. Luckily CRUD applications

represent the majority of web applications. To understand what AngularJS is good at, though, it helps to understand when an app is not a good fit for

AngularJS.

Games and GUI editors are examples of applications with intensive and tricky DOM manipulation. These kinds of apps are different from CRUD apps, and as a result are probably not a good fit for AngularJS. In these cases it may be better to use a library with a lower level of abstraction, such as jQuery.

**The Zen of AngularJS**

AngularJS is built around the belief that declarative code is better than imperative when it comes to building UIs and wiring software components together, while imperative code is excellent for expressing business logic.

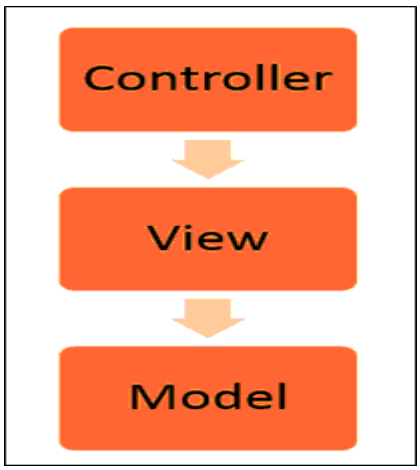
- It is a very good idea to decouple DOM manipulation from app logic. This dramatically improves the testability of the code.
- It is a really, *really* good idea to regard app testing as equal in importance to app writing. Testing difficulty is dramatically affected by the way the code is structured.
- It is an excellent idea to decouple the client side of an app from the server side. This allows development work to progress in parallel, and allows for reuse of both sides.
- It is very helpful indeed if the framework guides developers through the entire journey of building an app. From designing the UI, through writing the business logic, to testing.
- It is always good to make common tasks trivial and difficult tasks possible.

AngularJS frees you from the following pains.

- **Registering callbacks.** Registering callbacks clutters your code, making it hard to see the forest for the trees. Removing common boilerplate code such as callbacks is a good thing. It vastly reduces the amount of JavaScript coding *you* have to do, and it makes it easier to see what your application does.
- **Manipulating HTML DOM programmatically.** Manipulating HTML DOM is a cornerstone of AJAX applications, but it's cumbersome and error-prone. By declaratively describing how the UI should change as your application state changes, you are freed from low-level DOM manipulation tasks. Most applications written with AngularJS never have to programmatically manipulate the DOM, although you can if you want to.
- **Marshaling data to and from the UI.** CRUD operations make up the majority of AJAX applications' tasks. The flow of marshaling data from the server to an internal object to an HTML form, allowing users to modify the form, validating the form, displaying validation errors, returning to an internal model, and then back to the server, creates a lot of boilerplate code. AngularJS eliminates almost all of this boilerplate, leaving code that describes the overall flow of the application rather than all of the implementation details.
- **Writing tons of initialization code just to get started.** Typically you need to write a lot of plumbing just to get a basic "Hello World" AJAX app working. With AngularJS you can bootstrap your app easily using services, which are auto-injected into your application in a Guice-like dependency-injection style. This allows you to get started developing features quickly. As a bonus, you get full control over the initialization process in automated tests.

**AngularJS Architecture**

Angular.js follows the MVC architecture, the diagram of the MVC framework as shown below.




Angularjs Architecture Diagram

- The Controller represents the layer that has the business logic. User events trigger the functions which are stored inside your controller. The user events are part of the controller.
- Views are used to represent the presentation layer which is provided to the end users
- Models are used to represent your data. The data in your model can be as simple as just having primitive declarations. For example, if you are maintaining a student application, your data model could just have a student id and a name. Or it can also be complex by having a structured data model. If you are maintaining a car ownership application, you can have structures to define the vehicle itself in terms of its engine capacity, seating capacity, etc.

AngularJS starts automatically when the web page has loaded.

AngularJS extends HTML with **ng-directives**.

The **ng-app** directive defines an AngularJS application. The **ng-app** directive tells AngularJS that the  element is the "owner" of an AngularJS **application**.

The **ng-model** directive binds the value of the input field to the application variable **name**. The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

The **ng-bind** directive binds the content of the <p> element to the application variable **name**. The **ng-bind** directive binds application data to the HTML view.AngularJS expressions bind AngularJS data to HTML the same way as the **ng-bind** directive.

The **ng-app** directive defines the application, the **ng-controller** directive defines the controller.

The **ng-init** directive initializes AngularJS application variables.

AngularJS expressions are written inside double braces. **{{ expression }}**.AngularJS will "output" data exactly where the expression is written

### AngularJS Modules

A module defines the application functionality that is applied to the entire HTML page using the ng-app directive. It defines functionality, such as services, directives, and filters, in a way that makes it easy to reuse it in different applications.

### Creating a Module

A module is created by using the AngularJS function angular.module

```
<div ng-app="myApp">...</div>
  <script>
 var app = angular.module("myApp", []);
  </script>
```

**myApp.js**

var app = angular.module(**"myApp"**, []);

The [] parameter in the module definition can be used to define dependent modules.Without the [] parameter, you are not *creating* a new module, but *retrieving* an existing one.

### Architecture overview

Angular is a platform and framework for building client applications in HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.

The basic building blocks of an Angular application are *NgModules*, which provide a compilation context for *components*. NgModules collect related code into functional sets; an Angular app is defined by a set of NgModules. An app always has at least a *root module* that enables bootstrapping, and typically has many more *feature modules*.

- Components define *views*, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
- Components use *services*, which provide specific functionality not directly related to views. Service providers can be *injected* into components as *dependencies*, making your code modular, reusable, and efficient.

Both components and services are simply classes, with *decorators* that mark their type and provide metadata that tells Angular how to use them.

- The metadata for a component class associates it with a *template* that defines a view. A template combines ordinary HTML with Angular *directives* and *binding markup* that allow Angular to modify the HTML before rendering it for display.

- The metadata for a service class provides the information Angular needs to make it available to components through *dependency injection (DI)*.

An app's components typically define many views, arranged hierarchically. Angular provides the Router service to help you define navigation paths among views. The router provides sophisticated in-browser navigational capabilities.

**Modules**

Angular *NgModules* differ from and complement JavaScript (ES2015) modules. An NgModule declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities. An NgModule can associate its components with related code, such as services, to form functional units.

Every Angular app has a *root module*, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An app typically contains many functional modules.

Like JavaScript modules, NgModules can import functionality from other NgModules, and allow their own functionality to be exported and used by other NgModules. For example, to use the router service in your app, you import the Router NgModule.

Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability. In addition, this technique lets you take advantage of *lazy-loading*—that is, loading modules on demand—to minimize the amount of code that needs to be loaded at startup.

**Components**

Every Angular application has at least one component, the *root component* that connects a component hierarchy with the page document object model (DOM). Each component defines a class that contains application data and logic, and is associated with an HTML *template* that defines a view to be displayed in a target environment.

The @Component() decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

Decorators are functions that modify JavaScript classes. Angular defines a number of decorators that attach specific kinds of metadata to classes, so that the system knows what those classes mean and how they should work.

**Component Interaction**

***Pass data from parent to child with input binding.@Input. This decorator is used to **obtain data from the component**.
****Passing Data From Child to Parent (Using @Output) and Event Emitters.
@Output The attribute which is used when we want to send the data outside of the component.

This attribute is always combined with the event emitter which can be accessed outside of the component by using the event that is passed.

- The parent should have the provision to receive this message which will be assigned and used.
- The child should have the variable decorated with @output and a function which will emit this event. And the message which we want to send to the parent component.

EventEmitters
Event emitters are used in Angular to emit events in the components.
When we see the description of the above code we can see that we have used the @Output attribute and we have set it to the new EventEmitter, and we also have the method EmitValue which emits the value of the message property to the master component.

**Templates, directives, and data binding**

A template combines HTML with Angular markup that can modify HTML elements before they are displayed. Template *directives* provide program logic, and *binding markup* connects your application data and the DOM. There are two types of data binding.

- *Event binding* lets your app respond to user input in the target environment by updating your application data.
- *Property binding* lets you interpolate values that are computed from your application data into the HTML.

Before a view is displayed, Angular evaluates the directives and resolves the binding syntax in the template to modify the HTML elements and the DOM, according to your program data and logic. Angular supports *two-way data binding*, meaning that changes in the DOM, such as user choices, are also reflected in your program data.

Your templates can use *pipes* to improve the user experience by transforming values for display. For example, use pipes to display dates and currency values that are appropriate for a user's locale. Angular provides predefined pipes for common transformations, and you can also define your own pipes.

**Services and dependency injection**

For data or logic that isn't associated with a specific view, and that you want to share across components, you create a *service* class. A service class definition is immediately preceded by the @Injectable() decorator. The decorator provides the metadata that allows other providers to be **injected** as dependencies into your class.

*Dependency injection* (DI) lets you keep your component classes lean and efficient. They don't fetch data from the server, validate user input, or log directly to the console; they delegate such tasks to services.

**Routing**

The Angular Router NgModule provides a service that lets you define a navigation path among the different application states and view hierarchies in your app. It is modeled on the familiar browser navigation conventions.

- Enter a URL in the address bar and the browser navigates to a corresponding page.
- Click links on the page and the browser navigates to a new page.
- Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.

The router maps URL-like paths to views instead of pages. When a user performs an action, such as clicking a link, that would load a new page in the browser, the router intercepts the browser's behavior, and shows or hides view hierarchies.

If the router determines that the current application state requires particular functionality, and the module that defines it hasn't been loaded, the router can *lazy-load* the module on demand.

The router interprets a link URL according to your app's view navigation rules and data state. You can navigate to new views when the user clicks a button or selects from a drop box, or in response to some other stimulus from any source. The router logs activity in the browser's history, so the back and forward buttons work as well.

To define navigation rules, you associate *navigation paths* with your components. A path uses a URL-like syntax that integrates your program data, in much the same way that template syntax integrates your views with your program data. You can then apply program logic to choose which views to show or to hide, in response to user input and your own access rules.

- Together, a component and template define an Angular view.
- A decorator on a component class adds the metadata, including a pointer to the associated template.
- Directives and binding markup in a component's template modify views based on program data and logic.
- The dependency injector provides services to a component, such as the router service that lets you define navigation among views.

It then uses the Angular Language Service to read your tsconfig.json file, find all the templates you have in your application, and then provide language services for any templates that you open.

Language services include.

- Completions lists
- Quick info

AOT Diagnostic messages
Go to definition

**Component Styles**

Angular applications are styled with standard CSS. That means you can apply everything you know about CSS stylesheets, selectors, rules, and media queries directly to Angular applications.

Additionally, Angular can bundle *component styles* with components, enabling a more modular design than regular stylesheets.

This page describes how to load and apply these component styles.

**Using component styles**

For every Angular component you write, you may define not only an HTML template, but also the CSS styles that go with that template, specifying any selectors, rules, and media queries that you need.

One way to do this is to set the styles property in the component metadata. The styles property takes an array of strings that contain CSS code.

**Style scope**

The styles specified in @Component metadata *apply only within the template of that component*.

They are *not inherited* by any components nested within the template nor by any content projected into the component.

**Special selectors**

Component styles have a few special *selectors* from the world of shadow DOM style scoping.
**.host pseudo-class selector to target styles in the element that *hosts* the component (as opposed to targeting elements *inside* the component's template).
** host-context() pseudo-class selector, which works just like the function form of .host(). The .host-context() selector looks for a CSS class in any ancestor of the component host element, up to the document root. The .host-context() selector is useful when combined with another selector.

**Loading component styles**

There are several ways to add styles to a component.

- By setting styles or styleUrls metadata.
- Inline in the template HTML.
- With CSS imports. @import './hero-details-box.css';

Interpolation allows you to incorporate calculated strings into the text between HTML element tags and within attribute assignments. Template expressions are what you use to calculate those strings.

Interpolation is a technique that allows the user to bind a value to a UI element.

Interpolation binds the data one-way. This means that when value of the field bound using interpolation changes, it is updated in the page as well. It cannot change the *value* of the field. An object of the component class is used as data context for the template of the component. So the value to be bound on the view has to be assigned to a field in the component class.**Interpolation {{...}}**

**Property Binding**

Property binding is used to bind values to the DOM properties of the HTML elements. Like interpolation, property binding is a one-way binding technique. Property bindings are evaluated on every browser event and any changes made to the objects in the event, are applied to the properties.

DOM properties of the HTML elements shouldn’t be confused with HTML attributes of the elements. Every HTML element is represented as a JavaScript DOM object and every attribute of the HTML element is represented as a DOM property. For example, consider the following span element.

<span id="message"title="Message"style="font-style. italic; color. #FF0000;">This city is beautiful!

**Attribute directives**

Attribute directives manipulate the DOM by changing its behavior and appearance.

We use attribute directives to apply conditional style to elements, show or hide elements or dynamically change the behavior of a component according to a changing property.

**Structural directives**

These are specifically tailored to create and destroy DOM elements.

Some attribute directives — like hidden, which shows or hides an element — basically maintain the DOM as it is. But the structural Angular directives are much less DOM friendly, as they add or completely remove elements from the DOM. So, when using these, we have to be extra careful, since we’re actually changing the HTML structure.

****AngularJS lets you extend HTML with new attributes called Directives.AngularJS has a set of built-in directives which offers functionality to your applications.AngularJS also lets you define your own directives.
AngularJS directives are extended HTML attributes with the prefix ng-.The ng-app directive initializes an AngularJS application.The ng-init directive initializes application data.The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

***The application launches by bootstrapping the root AppModule, which is also referred to as an entryComponent. Among other things, the bootstrapping process creates the component(s) listed in the bootstrap array and inserts each one into the browser DOM.Each bootstrapped component is the base of its own tree of components. Inserting a bootstrapped component usually triggers a cascade of component creations that fill out that tree.While you can put more than one component tree on a host web page, most applications have only one component tree and bootstrap a single root component.This one root component is usually called AppComponent and is in the root module's bootstrap array.

**OnInit interface**

A lifecycle hook that is called after Angular has initialized all data-bound properties of a directive. Define an ngOnInit() method to handle any additional initialization tasks.

**ngOnInit()mode**_edit codeA callback method that is invoked immediately after the default change detector has checked the directive's data-bound properties for the first time, and before any of the view or content children have been checked. It is invoked only once when the directive is instantiated.

**NgFormdirective----**Creates a top-level FormGroup instance and binds it to a form to track aggregate form value and validation status.

As soon as you import the FormsModule, this directive becomes active by default on all <form> tags

**Data-binding** means communication between the TypeScript code of your component and your template which the user sees. Suppose, you have some business logic in your component TypeScript code to fetch some dynamic data from the server and want to display this dynamic data to the user via template because the user sees only the template. Here, we need some kind of binding between your TypeScript code and template (View). This is where data-binding comes into the picture in Angular because it is responsible for this communication.

Data-binding can be either one-way or two-way. Angular provides various types of data binding -

- **String Interpolation {{ data }}**
- **Property Binding**
  Property binding is also a one-way data binding, where we bind a property of a DOM element to a field which is a property we define in our component typescript code. Behind the scene, Angular converts string interpolation into property binding.

  For Example – <img [src]=”imgUrl” />

  However, we can use string interpolation here like <img src=”{{ imgUrl }}” /> , but property binding is always a lot cleaner and shorter syntax to bind image source.

- **Event Binding**
  Angular provides us with other types of binding, i.e., event binding, which is used to handle the events raised from the DOM like button click, mouse move etc. Let’s understand this with the help of an example –

  Suppose we have a button in the HTML template and we want to handle the click event of this button. To implement event binding, we will bind click event of a button with a method of the component.

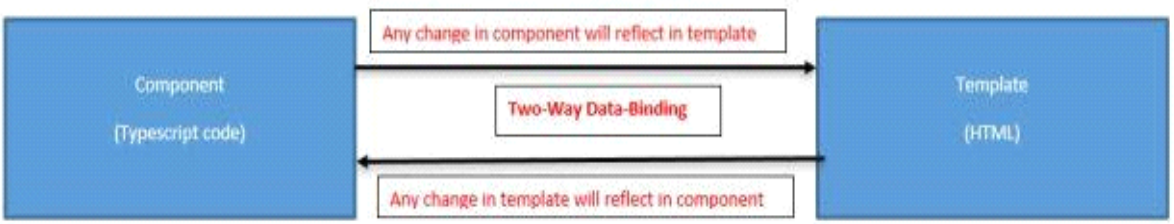
  **app.component.ts**

  export class AppComponent { onSave(){ console.log("Save operation is performed!") }

**Two-Way Data Binding**

Angular provides a very nice feature; i.e., two-way data binding. As of now, we have seen how to bind component data to view using one-way bindings. That means any change in the template(view) will not be reflected in the component typescript code.

Now, two-way binding has a feature to update data from component to view and vice-versa.



Syntax - For two-way data binding, we combine property binding and event binding both. Also, we can call “Banana in the Box”.

*[(ngModel)] = ”[property of your component]”*

**PIPE---** The | character is used to transform data. Following is the syntax for the same

It takes integers, strings, arrays, and date as input separated with **|** to be converted in the format as required and display the same in the browser.

Angular **6** provides some built-in pipes. The pipes are listed below –

Lowercasepipe Uppercasepipe Datepipe Currencypipe Jsonpipe Percentpipe Decimalpipe Slicepipe

To create a custom pipe, we have to import Pipe and Pipe Transform from Angular/core. In the @Pipe directive

The @Injectable() decorator has the providedIn metadata option, where you can specify the provider of the decorated service class with the root injector, or with the injector for a specific NgModule.

**DI** constructor(objLikeName. CLASS_NAME_FOR_OBJ_CRREATION)

**Why services**

Components shouldn't fetch or save data directly and they certainly shouldn't knowingly present fake data. They should focus on presenting data and delegate data access to a service. Services are a great way to share information among classes that *don't know each other*.
eg ng generate service hero

**@Injectable() services**

Notice that the new service imports the Angular Injectable symbol and annotates the class with the @Injectable() decorator. This marks the class as one that participates in the *dependency injection system*

**HttpClient**

Most front-end applications communicate with backend services over the HTTP protocol. Modern browsers support two different APIs for making HTTP requests. the XMLHttpRequest interface and the fetch() API.

The HttpClient in @angular/common/http offers a simplified client HTTP API for Angular applications that rests on the XMLHttpRequest interface exposed by browsers. Additional benefits of HttpClient include testability features, typed request and response objects, request and response interception, Observable apis, and streamlined error handling.

The sample app does not require a data server. It relies on the Angular *in-memory-web-api*, which replaces the *HttpClient* module's HttpBackend. The replacement service simulates the behavior of a REST-like backend.

**Router outlet**

The RouterOutlet is a directive from the router library that is used like a component. It acts as a placeholder that marks the spot in the template where the router should display the components for that outlet.

```
<router-outlet></router-outlet>
<!-- Routed components go here -->
```

The RouterLink directives on the anchor tags give the router control over those elements. The navigation paths are fixed, so you can assign a string to the routerLink (a "one-time" binding).

Had the navigation path been more dynamic, you could have bound to a template expression that returned an array of route link parameters (the *link parameters array*). The router resolves that array into a complete URL

```
<a routerLink="/heroes" routerLinkActive="active">Heroes</a>
```

The RouterLinkActive directive toggles css classes for active RouterLink bindings based on the current RouterState.

There more than two methods to navigate like navigate() , navigateByUrl(), and some other.. but we will mostly use these two.

- **navigate() .**

  Navigate based on the provided array of commands and a starting point. If no starting route is provided, the navigation is absolute.

  ```
  this.route.navigate(['/team/113/user/ganesh']);
  navigateByUrl()
  ```

Navigate based on the provided URL, which must be absolute.

```
this.route.navigateByUrl(['/team/113/user/ganesh']);
```

**navigateByUrl()** is similar to changing the location bar directly–we are providing the **whole** new URL.

**NgModules** configure the injector and the compiler and help organize related things together.

An NgModule is a class marked by the @NgModule decorator. @NgModule takes a metadata object that describes how to compile a component's template and how to create an injector at runtime. It identifies the module's own components, directives, and pipes, making some of them public, through the exports property, so that external components can use them. @NgModule can also add service providers to the application dependency injectors

**Web Application Testing - Techniques.**

1. **Functionality Testing**

Verify there is no dead page or invalid redirects. First check all the validations on each field.
Wrong inputs to perform negative testing. Verify the workflow of the system.
Verify the data integrity.

**Usability testing -** To verify how the application is easy to use with.

Test the navigation and controls. Content checking. Check for user intuition.

3. **Interface testing -** Performed to verify the interface and the dataflow from one system to other.

4. **Compatibility testing-** Compatibility testing is performed based on the context of the application.

- Browser compatibility Operating system compatibility
- Compatible to various devices like notebook, mobile, etc

5. **Performance testing -** Performed to verify the server response time and throughput under various load conditions.
Load testing Stress testing Soak testing Spike testing
6. **Security testing -** Performed to verify if the application is secured on web as data theft and unauthorized access are more common issues and below are some of the techniques to verify the security level of the system.

**JavaScript Unit Testing Frameworks**

1. Unit.js. 2. QUnit. 3. Jasmine. 4. Karma. 5. Mocha. 6. Jest. 7. AVA.

**AngularJS Unit Testing.**

Karma Jasmine angular-mocks