Javascript OOPS

Jquery

AngularJS (directives)

Ionicframework

Cordova

Grunt

https://www.codementor.io/angularjs/tutorial/angularjs-interview-questions-sample-answers

http://career.guru99.com/top-25-angular-js-interview-questions/

https://www.toptal.com/angular-js/interview-questions

1. SQL - DB - Normalization

2. Banking Services security

3. SQL - JOIN Command

4. XMTP, SOAP, REST Protocol

5. Push Notification - Cordova

6. Store objects- Plist, Sqllite, NS..

**#################### ----- Java Script -------- ####################**

1. **What are the types in javascript?**

var length = 16;                               // Number  
var lastName = "Johnson";                      // String  
var cars = ["Saab", "Volvo", "BMW"];           // Array  
var x = {firstName:"John", lastName:"Doe"};    // Object

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

var x = 16 + 4 + "Volvo";

Result : - 20Volvo

In JavaScript, there are three primary data types, two composite data types, and two special data types.

[Primary Data Types](javascript:void(0))

The primary (primitive) data types are:

* String
* Number
* Boolean

[Composite Data Types](javascript:void(0))

The composite (reference) data types are:

* Object
* Array

[Special Data Types](javascript:void(0))

The special data types are:

* Null
* Undefined

**Q. How to get length of Object**

**var obj={**

**name:"Gajanan",**

**address:"Hyderabad"**

**}**

**Object.keys(obj).length**

**2**

**Q. Diff between Map & Filters**

### filter()

The filter() method creates an array filled with all array elements that pass a test implemented by the provided function. The filter method is well suited for particular instances where the user must identify certain items in an array that share a common characteristic. For example, consider the following array:

var playersArr = [  
 {name: 'Jason', footedness: 'left', position: 'forward'},  
 {name: 'Blake', footedness: 'right', position: 'defense'},  
 {name: 'Philip', footedness: 'right', position: 'goalie'},  
 {name: 'Logan', footedness: 'left', position: 'defense'},  
 {name: 'Will', footedness: 'right', position: 'forward'}  
];

Let’s find all of the players in the array that are left-footed using the filter() method!

var leftFootArr = playersArr.filter(function(player){  
 return player.footedness === 'left';  
});

console.log(leftFootArr);

/\*This will log: [{name: 'Jason', footedness: 'left', position: 'forward'}, {name: 'Logan', footedness: 'left', position: 'defense'}];\*/

### map()

The map() method creates a new array with the results of calling a function for every array element. The map method allows items in an array to be manipulated to the user’s preference, returning the conclusion of the chosen manipulation in an entirely new array. For example, consider the following array:

var agesArr = [25, 36, 49, 64, 81];

Let’s take the ages inside of the array and find their respective square roots.

function root() {  
 var roots = agesArr.map(Math.sqrt);  
 return roots;  
};

root();

//This will return: [ 5, 6, 7, 8, 9 ];

**JS Variable declaration & accessable**

**function Person(name) {**

**this.name = "ppppppp" // Public- can access with using obj**

**var a="ggggggggggg"; // Private – canot access with obj**

**b="bbbbb"; //Global – window.b**

**};**

**Person.abc = “text”; // static variable**

**var pr = new Person();**

**pr.name**

**"ppppppp"**

**pr.b**

**undefined**

**b**

**"bbbbb"**

**pr.a**

**undefined**

**Q. what is function invoking in javascript?**

## Invoking a JavaScript Function

Some people use the term "call a function" instead of "invoke a function".It is also quite common to say "call upon a function", "start a function", or "execute a function".

e.g

function myFunction(a, b) {  
    return a \* b;  
}  
myFunction(10, 2);           // myFunction(10, 2) will return 20  
window.myFunction(10, 2);    // window.myFunction(10, 2) will also return 20

## Invoking a Function as a Method

var myObject = {  
    firstName:"John",  
    lastName: "Doe",  
    fullName: function () {  
        return this.firstName + " " + this.lastName;  
    }  
}  
myObject.fullName();         // Will return "John Doe"

## Invoking a Function with a Function Constructor

If a function invocation is preceded with the **new** keyword, it is a constructor invocation.

// This is a function constructor:  
function myFunction(arg1, arg2) {  
    this.firstName = arg1;  
    this.lastName  = arg2;  
}  
  
// This creates a new object  
var x = new myFunction("John","Doe");  
x.firstName;                             // Will return "John"

**Q. Difference between Array concat & join method**

The **join()** method creates and returns a new string by concatenating all of the elements in an array (or an [array-like object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Indexed_collections#Working_with_array-like_objects)), separated by commas or a specified separator string.

var elements = ['Fire', 'Wind', 'Rain'];

console.log(elements.join());

// expected output: Fire,Wind,Rain

console.log(elements.join(''));

// expected output: FireWindRain

console.log(elements.join('-'));

// expected output: Fire-Wind-Rain

The **concat()** method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

var array1 = ['a', 'b', 'c'];

var array2 = ['d', 'e', 'f'];

console.log(array1.concat(array2));

// expected output: Array ["a", "b", "c", "d", "e", "f"]

**Q. Remove Duplicate elements from array**

<script>

let dupArr = [1, 2, 0, 1, 2, 0, 1, 3, 0, 1, 3, 2];

let dupArrObj = [{ id: 1, value: 'a' }, { id: 2, value: 'b' }, { id: 1, value: 'c' }];

function onLoadBody() {

let filterArr = [];

let filterArrObj = [];

// --- ES5 ---

dupArr.forEach(item => {

if (filterArr.indexOf(item) < 0) {

filterArr.push(item);

}

});

dupArrObj.forEach(item => {

let indx = filterArrObj.findIndex((element) => {

return element.id === item.id;

});

if (indx < 0) {

filterArrObj.push(item);

}

});

// --- ES6 ---

filterArr = [...new Set(dupArr)];

filterArrObj = [...new Set(dupArrObj.map(({ id }) => id))];

document.getElementById('arrDiv').innerHTML = filterArr.sort();

document.getElementById('arrDivObj').innerHTML = JSON.stringify(filterArrObj);

}

</script>

**Q. Get 2 smallest number from number type variable & add that 2 numbers**

let num = 7825461;

let arrSplt = String(num).split("").map(Number).sort();

let addDigit = arrSplt[0] + arrSplt[1];

document.getElementById('arrSum').innerHTML = addDigit;

1. **What call,apply and bind in javascript?**

## Invoking a Function with a Function Method

In JavaScript, functions are objects. JavaScript functions have properties and methods.

**call()** and **apply()** are predefined JavaScript function methods. Both methods can be used to invoke a function, and both methods must have the owner object as first parameter

function myFunction(a, b) {  
    return a \* b;  
}  
myObject = myFunction.call(myObject, 10, 2);     // Will return 20

function myFunction(a, b) {  
    return a \* b;  
}  
myArray = [10, 2];  
myObject = myFunction.apply(myObject, myArray);  // Will also return 20

Both methods takes an owner object as the first argument. The only difference is that call() takes the function arguments separately, and apply() takes the function arguments in an array. In JavaScript strict mode, the first argument becomes the value of **this** in the invoked function, even if the argument is not an object. In "non-strict" mode, if the value of the first argument is null or undefined, it is replaced with the global object. With call() or apply() you can set the value of **this**, and invoke a function as a new method of an existing object.

**Bind Function**

var checkNumericRange = function (value) {

if (typeof value !== 'number')

return false;

else

return value >= this.minimum && value <= this.maximum;

}

// The range object will become the this value in the callback function.

var range = { minimum: 10, maximum: 20 };

// Bind the checkNumericRange function.

var boundCheckNumericRange = checkNumericRange.bind(range);

// Use the new function to check whether 12 is in the numeric range.

var result = boundCheckNumericRange (12);

document.write(result);

// Output: true

var data;

var dataReadyEvent = document.createEvent("Event");

dataReadyEvent.initEvent("dataReady", true, false);

function DataObject() {

this.name = "Data Object";

this.data = function () {

return data;

}

this.onDataCompleted = dataReadyHandler;

document.addEventListener('dataReady', this.onDataCompleted.bind(this));

// To see the result of not using bind, comment out the preceding line,

// and uncomment the following line of code.

// document.addEventListener('dataReady', this.onDataCompleted);

}

function dataReadyHandler() {

if (console && console.log) {

console.log("Data object property value: " + this.name);

console.log("Data object property value: " + this.data());

}

}

setTimeout(function () {

data = [0, 1, 2, 3];

document.dispatchEvent(dataReadyEvent);

}, 5000);

}

var dataObj = new DataObject();

function ArgTest(a, b){

var s = "";

s += "Expected Arguments: " + ArgTest.length;

s += "<br />";

s += "Passed Arguments: " + arguments.length;

return s;

}

document.write(ArgTest(1, 2));

// Output:

// Expected Arguments: 2

// Passed Arguments: 2

var randomNum = ((Math.random () \* 2 | 0) + 1) - 1; // random number between 0 and 1​

1. **How to create the object in javascript?**

var me={ fname : "Gajanan", lname : "Pawale", colors : ['red', 'yellow', 'green', 'blue'] };

for(var key in me){

if(me.hasOwnProperty(key)){

if(me instanceof Array) {

for(var item in me[key]){

console.log(me[key]);

}

} else

console.log(me[key]);

}

}

1. What does the typeof and instanceof keyword in javascript?

// Boolean

var str3 = true ;

console.log (str3);

console.log (str3 instanceof Boolean); // false: expect true

typeof str3 // "boolean"

console.log(typeof str3 === "boolean" ); // true

// Number

var str4 = 100 ;

console.log (str4);

console.log (str4 instanceof Number); // false: expect true

console.log (typeof str4 == "number" ); // true

function Dog() {}

var obj = new Dog;

typeof obj == 'Dog' // false, typeof obj is actually "object"

obj instanceof Dog // true, what we want in this case

var str = 'hello word';

var arr=[];

console.log(arr instanceof Array); // true

console.log(typeof str) // string

console.log(typeof arr) // object

1. **What is reference type in javascript?**

undefined, null, number, string, boolean and object of which only object is a "reference" type.

The ECMAScript language types are Undefined, Null, Boolean, String, Number, and Object

The only "reference" type is the Object.

function isReferenceType( value ) {

return Object(value) === value;

}

function isPrimitiveType( value ) {

return Object(value) !== value;

}

1. What is scope in javascript?

In JavaScript, objects and functions are also variables.

**In JavaScript, scope is the set of variables, objects, and functions you have access to.**

JavaScript has function scope: The scope changes inside functions.

## Local JavaScript Variables

Variables declared within a JavaScript function, become **LOCAL** to the function. Local variables have **local scope**: They can only be accessed within the function.

// code here can not use carName  
  
function myFunction() {  
    var carName = "Volvo";  
  
    // code here can use carName  
  
}

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions. Local variables are created when a function starts, and deleted when the function is completed.

## Global JavaScript Variables

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.

var carName = " Volvo";  
  
// code here can use carName  
  
function myFunction() {  
  
    // code here can use carName   
  
}

## The Lifetime of JavaScript Variables

The lifetime of a JavaScript variable starts when it is declared. Local variables are deleted when the function is completed. Global variables are deleted when you close the page.

## **Q. Variable Lifetime**

Global variables live as long as your application (your window / your web page) lives.

Local variables have short lives. They are created when the function is invoked, and deleted when the function is finished.

1. **What is hoisting in javascript?**

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).

In JavaScript, a variable can be declared after it has been used. In other words; a variable can be used before it has been declared.

var x; // Declare x  
x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x;

## **JavaScript Initializations are Not Hoisted**

JavaScript only hoists declarations, not initializations.

var x = 5; // Initialize x  
var y;     // Declare y  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x + " " + y;           // Display x and y

//output –5 undefined

y = 7;    // Assign 7 to y

 var myvar = 'my value';

(function() {

  alert(myvar); // undefined

  var myvar = 'local value';

})();

 var myvar = 'my value';

(function() {

  alert(myvar); // my value

  myvar = 'local value';

})();

//hoisting JS function

function foo() {

// A function expression

var bar = function() {

return 3;

};

return bar();

// The variable bar already exists, and this code will never be reached

var bar = function() {

return 8;

};

}

var testvar = foo();

console.log(testvar); //3

function foo1() {

// A function declaration

function bar() {

return 3;

}

return bar();

// This function declaration will be hoisted and overwrite the previous one function

function bar() {

return 8;

}

}

var testvar1 = foo1();

console.log(JSON.stringify(testvar1)); //8

1. **What is clousure function in javascript?**

A **closure** is an inner **function** that has access to the outer (enclosing)**function's** variables—scope chain. The **closure** has three scope chains: it has access to its own scope (variables defined between its curly brackets), it has access to the outer **function's** variables, and it has access to the global variables.

<http://www.masterjavascript.io/blog/2016/04/24/understanding-closures/>

var add = (function () {  
    var counter = 0;  
    return function () {return counter += 1;}  
})();  
  
add();  
add();  
add();

//output --- 3

## **Example Explained**

The variable **add** is assigned the return value of a self-invoking function. 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.

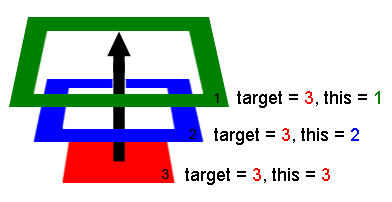
A closure is a function having access to the parent scope, even after the parent function has closed.

|  |
| --- |
| function showName (firstName, lastName) { |
|  | ​var nameIntro = "Your name is "; |
|  | // this inner function has access to the outer function's variables, including the parameter​ |
|  | ​function makeFullName () { |
|  | ​return nameIntro + firstName + " " + lastName; |
|  | } |
|  | ​ |
|  | ​return makeFullName (); |
|  | } |
|  | ​ |
|  | showName ("Michael", "Jackson"); // Your name is Michael Jackson |

1. **What is event boubling in javascript?**

|  |
| --- |
| <html> |
|  | <body> |
|  | <link type="text/css" rel="stylesheet" href="[example.css](http://javascript.info/files/tutorial/browser/events/bubbling/bubble/example.css)"> |
|  | <div class="d1" onclick="highlight(this)">1 |
|  | <div class="d2" onclick="highlight(this)">2 |
|  | <div class="d3" onclick="highlight(this)">3 |
|  | </div> |
|  | </div> |
|  | </div> |
|  | <script> |
|  | function highlight(elem) { |
|  | elem.style.backgroundColor='yellow' |
|  | alert(elem.className) |
|  | elem.style.backgroundColor = '' |
|  | } |
|  | </script> |
|  | </body> |
|  | </html> |

.d1{background-color:green;position:relative;width:150px;height:150px;text-align:center;cursor:pointer;}.d2{background-color:blue;position:absolute;top:25px;left:25px;width:100px;height:100px;}.d3{background-color:red;position:absolute;top:25px;left:25px;width:50px;height:50px;line-height:50px;}



<!DOCTYPE html>

<html>

<body>

<p>Click the button to find out if the onclick event is a bubbling event.</p>

<button onclick="myFunction(event)">Try it</button>

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

<script>

function myFunction(event) {

var x = event.bubbles;

document.getElementById("demo").innerHTML = x;

}

</script>

</body>

</html>

1. **Give any example of recursion in javascript?**

The factorial of six is:

6 × 5 × 4 × 3 × 2 × 1 = 720

var factorial = function(number) {

if (number <= 0) { // terminal case

return 1;

} else { // block to execute

return (number \* factorial(number - 1));

}

};

console.log(factorial(6));

// 720

1. **What is prototype in javascript?**

Every JavaScript object has a prototype. The prototype is also an object. All JavaScript objects inherit their properties and methods from their prototype.

## JavaScript Prototypes

All JavaScript objects inherit the properties and methods from their prototype. Objects created using an object literal, or with new Object(), inherit from a prototype called Object.prototype. Objects created with new Date() inherit the Date.prototype. The Object.prototype is on the top of the prototype chain. All JavaScript objects (Date, Array, RegExp, Function, ....) inherit from the Object.prototype.

## **Adding Properties to a Prototype**

You cannot add a new property to a prototype the same way as you add a new property to an existing object, because the prototype is not an existing object.

function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}

var myFather = new person("John", "Doe", 50, "blue");  
var myMother = new person("Sally", "Rally", 48, "green");

person.nationality = "English";

To add a new property to a constructor, you must add it to the constructor function:

function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
    this.nationality = "English"  
}

or

## **Using the prototype Property**

The JavaScript prototype property allows you to add new properties to an existing prototype:

function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}  
person.prototype.nationality = "English";

1. How achive the inheritance and encapsulation in javascript?
2. Is JavaScript call-by-value or call-by-reference?

JavaScript is call-by-value.

1. Garbage Collection

Instead of requiring manual deallocation, JavaScript relies on a technique called garbage collection. The JavaScript interpreter is able to detect when an object will never again be used by the program. When it determines that an object is unreachable (i.e., there is no longer any way to refer to it using the variables in the program), it knows that the object is no longer needed and its memory can be reclaimed. Consider the following lines of code, for example:

var s = "hello"; // Allocate memory for a string

var u = s.toUpperCase( ); // Create a new string

s = u; // Overwrite reference to original string

After this code runs, the original string "hello" is no longer reachable -- there are no references to it in any variables in the program. The system detects this fact and frees up its storage space for reuse.

Garbage collection is automatic and is invisible to the programmer. You can create all the garbage objects you want, and the system will clean up after you! You need to know only enough about garbage collection to trust that it works; you don't have to wonder about where all the old objects go

## **JavaScript Objects are Mutable**

Objects are mutable: They are addressed by reference, not by value. If y is an object, the following statement will not create a copy of y: The object x is not a **copy** of y. It **is** y. Both x and y points to the same object. Any changes to y will also change x, because x and y are the same object.

var x = y;  // This will not create a copy of y.

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}  
  
var x = person;  
x.age = 10;           // This will change both x.age and person.age

1. **Diff Between undefined & null**

🡪

|  |  |
| --- | --- |
|  | In JavaScript, undefined means a variable has been declared but has not yet been assigned a value, such as:  var TestVar;  alert(TestVar); //shows undefined  alert(typeof TestVar); //shows undefined  null is an assignment value. It can be assigned to a variable as a representation of no value:  var TestVar = null;  alert(TestVar); //shows null  alert(typeof TestVar); //shows object  From the preceding examples, it is clear that undefined and null are two distinct types: undefined is a type itself (undefined) while null is an object.  null === undefined // false  null == undefined // true  null === null // true  and  null = 'value' // ReferenceError  undefined = 'value' // 'value' **Q. Falsy values** The following values evaluate to false (also known as [Falsy](https://developer.mozilla.org/en-US/docs/Glossary/Falsy" \o "Falsy: A falsy value is a value that translates to false when evaluated in a Boolean context.) values):   * false * undefined * null * 0 * NaN * the empty string ("")   e.g :  function abc(){  **if(!null && !undefined){**  return "abc func"; }  }  abc()//"abc func" --- true  function abc(){  **if(!0 && !undefined){**  return "abc func"; }  }  undefined  abc() //"abc func" --- true  function abc(){  **if(0 && !undefined){**  return "abc func"; }  }  abc() //undefined --- false   1. Object inheritance   var x = { this.name="Gajanan"; }  var y = Object.create(x);  y  Function\_\_  proto\_\_: ()  arguments: null  caller: null  length: 0  name: "x"  prototype: Object\_\_  proto\_\_: ()  [[FunctionLocation]]: VM221:1  [[Scopes]]: Scopes[1]  function x(){ this.name = "gggg"; }  undefined  var y = Object.create(x);  undefined  y  Function {}  \_\_proto\_\_: x()  arguments: null  caller: null  length: 0  name: "x"  prototype: Object  y.name  "x"  var x = {a:1,b:2};  var y = Object.create(x);  x //Object {a: 1, b: 2}  y // Object {} // below as shown property  Object  \_\_proto\_\_: Object  a: 1  b: 2  \_\_proto\_\_: Object  check inheritance  y.a  1  y.b  2   1. Function Overloading (JS do not support function overloading)   e.g  function add(){ return 1; }  function add(a,b){ return a+b; }  function add(a,b,c){ return a+b+c; }  function add(a,b,c,d){ return a+b+c; }  function add(a,b,c,d){ return a+b+c+d; }  add(); // NaN  add(1,2); // NaN  add(1,2,3); // NaN  add(1,2,3,”4”);//10  function add(a,b){ return a+b; }  function add(a,b,c){ return a+b+c; }  function add(a,b,c,d){ return a+b+c; }  function add(a,b,c,d){ return a+b+c+d; }  function add(){ return 1; }  undefined  add(1,2) //1  add(1,2,3,5) //1   1. **Array length**   e.g  var arr1 = ["a","b","c"];  arr1.length // 3  arr1[4] = "e"  arr1.length //5  arr1  ["a", "b", "c", undefined × 1, "e"]  arr1[3] = "d"  arr1  ["a", "b", "c", "d", "e"]  arr1[10] = "ggg";  "ggg"  arr1  ["a", "b", "c", "d", "e", undefined × 5, "ggg"]  4. HTML alert window execution  **html file**  <!DOCTYPE html>  <html>  <head>  <title>Page Title</title>  <script src="overview.js" type="text/javascript">  alert("Header");  console.log("Header Alert won't come --- Output - overview then body");  </script>  </head>  <body>  <script type="text/javascript">  alert("Body");  </script>  <h1>This is a Body</h1>  </body>  </html>  **overview.js**  alert(“Overview”);  O/p -----  Overview  Body |
|  | **If you remove src overview.js from head tag then output**  Header  Body |
|  | **Q**. **Type Conversation (left to right execution)**  "1"+2+3  "123"  1+undefined  NaN  undefined+1  NaN  undefined+"1"  "undefined1"  null+1  1  1+null  1  1+2+"3"  "33"  The typeof Operator  You can use the **typeof** operator to find the data type of a JavaScript variable.  typeof "John"                 // Returns "string"  typeof 3.14                   // Returns "number" typeof NaN                    // Returns "number" typeof false                  // Returns "boolean" typeof [1,2,3,4]              // Returns "object" typeof {name:'John', age:34}  // Returns "object" typeof new Date()             // Returns "object" typeof function () {}         // Returns "function" typeof myCar                  // Returns "undefined" \* typeof null                   // Returns "object"  Please observe:   * The data type of NaN is number * The data type of an array is object * The data type of a date is object * The data type of null is object * The data type of an undefined variable is **undefined** \* * The data type of a variable that has not been assigned a value is also **undefined** \*   The **constructor** property returns the constructor function for all JavaScript variables.  "John".constructor                 // Returns "function String()  { [native code] }" (3.14).constructor                 // Returns "function Number()  { [native code] }" false.constructor                  // Returns "function Boolean() { [native code] }" [1,2,3,4].constructor              // Returns "function Array()   { [native code] }" {name:'John', age:34}.constructor  // Returns" function Object()  { [native code] }" new Date().constructor             // Returns "function Date()    { [native code] }" function () {}.constructor         // Returns "function Function(){ [native code] }" |

1. Object copy ref

var obj = { a:1, b:2 };

var obj1 = {c:3};

var obj2 = obj1 = obj;

obj2.a = 11; obj2.b = 12; obj2.c = 13;

obj.a //11

obj1.a //11

obj1.c //13

obj1.a = 21; obj1.b = 22; obj1.c=23;

obj.a //21

obj2.a //21

function abc(){

console.log(this);

}

abc()

output ----

Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, frames: Window, …}

|  |  |
| --- | --- |
|  | **Q. Javascript is always a synchronous(blocking) single thread language but we can make Javascript act Asynchronous through programming.**  Synchronous code:  console.log('a');  console.log('b');  Asynchronous code:  console.log('a');  setTimeout(function() {  console.log('b');  }, 1000);  setTimeout(function() {  console.log('c');  }, 1000);  setTimeout(function() {  console.log('d');  }, 1000);  console.log('e');  This outputs: a e b c d |

1. What is localStorage and sessionStorage in HTML5?

localStorage and sessionStorage both extend Storage. There is no difference between them except for the intended "non-persistence" of sessionStorage.

That is, the data stored in localStorage persists until explicitly deleted. Changes made are saved and available for all current and future visits to the site.

For sessionStorage, changes are only available per window (or tab in browsers like Chrome and Firefox). Changes made are saved and available for the current page, as well as future visits to the site **on the same window**. Once the window is closed, the storage is deleted.

localStorage - use for long term use.  
sessionStorage - use when you need to store something that changes or something temporary

|  |  |
| --- | --- |
|  | This is an extremely broad scope question, and a lot of the pros/cons will be contextual to the situation.  In all cases these storage mechanisms will be specific to an individual browser on an individual computer/device. Any requirement to store data on an ongoing basis across sessions will need to involve your application server side - most likely using a database, but possibly XML or a text/CSV file.  localStorage, sessionStorage and cookies are all client storage solutions. Session data is held on the server where it remains under your direct control. localStorage and sessionStorage localStorage and sessionStorage are relatively new APIs (meaning not all legacy browsers will support them) and are near identical (both in APIs and capabilities) with the sole exception of persistence. sessionStorage (as the name persists) is only available for the duration of the browser session (and is deleted when the window is closed) - it does however survive page reloads (source [DOM Storage guide - Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Storage)).  Clearly, if the data you are storing needs to be available on an ongoing basis then localStorage is preferable to sessionStorage - although you should note both can be cleared by the user so you should not rely on the continuing existence of data in either case.  localStorage and sessionStorage are perfect for persisting non-sensitive data needed within client scripts between pages (for example: preferences, scores in games). The data stored in localStorage and sessionStorage can easily be read or changed from within the client/browser so should not be relied upon for storage of sensitive or security related data within applications. |

# localStorage vs. sessionStorage vs. Cookies

In terms of capabilities, cookies only allow you to store strings. sessionStorage and localStorage allow you to store JavaScript primitives but not Objects or Arrays (it is possible to JSON serialise them to store them using the APIs). Session storage will generally allow you to store any primitives or objects supported by your Server Side language/framework.

**Q. What is javascript hoisting (Refer:** <http://www.programmerinterview.com/index.php/javascript/javascript-hoisting/>)?

Ans) Because Javascript has Function Scope, this also means that variables declared within a function are visible anywhere

**Q. How do you check if javascript is enabled or not?**

using the <noscript> element

What are the different ways we can create objects in javascript?

Using an Object Literal

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Using the JavaScript Keyword new

The following example also creates a new JavaScript object with four properties:

Example

var person = new Object();

person.firstName = "John";

person.lastName = "Doe";

person.age = 50;

person.eyeColor = "blue";

Using an Object Constructor

The examples above are limited in many situations. They only create a single object.

Sometimes we like to have an "object type" that can be used to create many objects of one type.

The standard way to create an "object type" is to use an object constructor function:

Example

function person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eye;

}

var myFather = new person("John", "Doe", 50, "blue");

var myMother = new person("Sally", "Rally", 48, "green");

How to write Jquery plugins?

$.fn.greenify = function() {

this.css( "color", "green" );

};

$( "a" ).greenify(); // Makes all the links green.

What is jsonp?

Cross domain reqs

How do you access cross domain url using ajax?

Ans) Using JSONP or CORS

Is there a way to cancel a ajax request? If so How can you cancel an ajax request ?

Ans) The $.ajax() method returns an object that could be used to do that:

var xhr = $.ajax( url: "http://someurl.org", success: function() { ... } );

and then later when you want to stop the request:

xhr.abort();

ajax can be stopped from being initiated, but once its made its not possible to stop it.

**Q. Abort api request**

you can abort the AJAX request in the browser but of course if it has already hit the web server, that's a whole different story. The web server will continue to execute the request. It's just that since the browser aborted the request the web server will send the response to the void.

so it means that if an already initiated request is aborted using the .abort method it will not be catered for at the client side but that will depend when you call the abort method.

1. // abort function with check readystate
2. function abortAjax(xhr) {
3. if(xhr && xhr.readystate != 4){
4. xhr.abort();
5. }
6. }
7. // this function usage
8. abortAjax(xhr);

What are the different object oriented constructs supported by javascript?

If javascript is disbled will the HTML5 api's work ?

How do you identify memory leaks in javascript?

<https://auth0.com/blog/four-types-of-leaks-in-your-javascript-code-and-how-to-get-rid-of-them/>

**Q. How to avoid polluting Global Namespace in Javascript?**

<http://lillylabs.no/2014/09/19/avoid-polluting-the-global-namespace-in-javascript/>

What is "use strict" in ECMA Script 5 ?

**"use strict"** is a declaration that tells the interpreter to interpret the code in “strict mode”.

**Q. What is global namespace  in javascript?**

**Ans)** The global namespace is a namespace that is common to all JavaScript code in your JavaScript interpreter. It’s also known as “global scope”.

The global namespace is the window object. You don't have to declare a variable using the var keyword; as soon as you start to use it, and the variable hasn't already been defined, it is simply appended to the global namespace. Hence, x = 2; and window.x = 2; are equivalent.

Why is it bad idea to have vars/functions on a global level?

**Ans)** Because if you're adding lots of 3rd party libraries/ scripts, they all share the same global object(which is window object), there's the chance of name collisions. This is a real life problem with all the libraries which use $ as an alias (jQuery, Prototype and more).

**Q. How to delete the table row in javascript**

<!DOCTYPE html>

<html>

<head>

<style>

table, td {

border: 1px solid black;

}

</style>

</head>

<body>

<table id="myTable">

<tr>

<td>Row 1</td>

<td><input type="button" value="Delete" onclick="deleteRow(this)"></td>

</tr>

<tr>

<td>Row 2</td>

<td><input type="button" value="Delete" onclick="deleteRow(this)"></td>

</tr>

<tr>

<td>Row 3</td>

<td><input type="button" value="Delete" onclick="deleteRow(this)"></td>

</tr>

</table>

<script>

function deleteRow(r) {

var i = r.parentNode.parentNode.rowIndex;

document.getElementById("myTable").deleteRow(i);

}

</script>

</body>

</html>

**Q. variable scope, Difference between var and let in JavaScript**

Var a = 10;

Window.a

10

let a =10 ;

window.a;

undefined

* difference between them is that var is function scoped and let is block scoped.

**#################### ----- JQUERY -------- ####################**

1. **How to avoid conflict $**

-🡪 <script>

$.noConflict();

jQuery( document ).ready(function( $ ) {

// Code that uses jQuery's $ can follow here.

});

// Code that uses other library's $ can follow here.

</script>

var jq = $.noConflict();  
jq(document).ready(function(){  
    jq("button").click(function(){  
        jq("p").text("jQuery is still working!");  
    });  
});

**Q. What is jQuery delegate method?**

**Ans)** The delegate() method attaches one or more event handlers for specified elements that are children of selected elements, and specifies a function to run when the events occur. Event handlers attached using the delegate() method will work for both current and FUTURE elements (like a new element created by a script).

ex) When a <p> element inside a <div> element is clicked, change the background color of all <p> elements:

$("div").delegate("p", "click", function(){

   $("p").css("background-color", "pink");

});

**Q. What is jQuery live method?**

**Ans)** The live() method was deprecated in jQuery version 1.7, and removed in version 1.9. Use the on() method instead.

The live() method attaches one or more event handlers for selected elements, and specifies a function to run when the events occur.

Event handlers attached using the live() method will work for both current and FUTURE elements matching the selector (like a new element created by a script).

ex) $("button").live("click", function(){

$("p").slideToggle();

});

**Q. How do you show/hide a div when window scroll bar to some extent lets say 200px?**

Ans) Make sure you have a long content

$(document).on("scroll", function(){

 var scrollTop = $(document).scrollTop();

if(scrollTop > 100) {

      $("#myDiv").show("slow");

} else {

       $("#myDiv").hide("slow");

}

});

<div id="myDiv" style="width: 200px; height:200px; border-style:solid; border-color:red; margin:auto;position: fixed; top: 100px;display: none">

  <span>Hello I am Bhasker</span>

</div>

1. **How to access middle child of ul using jquery**

<ul>

<li>Glen</li>

<li>Tane</li>

<li>Ralph</li>

<li>David</li>

</ul>

<script>

$( "ul li:nth-child(2)" ).append( "<span> - 2nd!</span>" );

</script>

* Glen
* Tane - 2nd!
* Ralph
* David
  1. **What is bind, live & delegate methods**

.bind() attacheds events to elements that exist or match the selector at the time the call is made. Any elements created afterwards or that match going forward because the class was changed, will not fire the bound event.

.live() works for existing and future matching elements. Before jQuery 1.4 this was limited to the following events: click, dblclick mousedown, mouseup, mousemove, mouseover, mouseout, keydown, keypress, keyup

In short: .bind() will only apply to the items you currently have selected in your jQuery object. .live() will apply to all current matching elements, as well as any you might add in the future.

1. **what is polyfill**

**🡪** A polyfill is a browser fallback, made in JavaScript, that allows functionality you expect to work in modern browsers to work in older browsers, e.g., to support canvas (an HTML5 feature) in older browsers.

1. Which is fastest jQuery Selector?
2. Which is Slowest jQuery Selctor?
3. How Call the ajax in jQuery?
4. What are ajax function arguments in jQuery?
5. What is function chaining in jQuery?

Can I create a jquery object with Jquery?

**################# ----- HTML ------ #############**

1. **New Features in HTML5**

Key new features of HTML5 include:

* Improved support for embedding graphics, audio, and video content via the new [<canvas>](http://www.w3schools.com/tags/tag_canvas.asp), [<audio>](http://www.w3schools.com/tags/tag_audio.asp), and [<video>](http://www.w3schools.com/tags/tag_video.asp) tags.
* Extensions to the JavaScript API such as [geolocation](http://www.w3schools.com/html/html5_geolocation.asp) and [drag-and-drop](http://www.w3schools.com/html/html5_draganddrop.asp) as well for [storage](http://www.w3schools.com/html/html5_webstorage.asp) and [caching](http://www.w3schools.com/html/html5_app_cache.asp).
* Introduction of [“web workers”](http://www.w3schools.com/html/html5_webworkers.asp).
* Several new semantic tags were also added to complement the structural logic of modern web applications. These include the <main>, <nav>, <article>, <section>, <header>, <footer>, and <aside> tags.
* New form controls, such as <calendar>, <date>, <time>, <email>, <url>, and <search>.
* [Web workers](http://www.w3schools.com/html/html5_webworkers.asp) at long last bring multi-threading to JavaScript.
* A web worker is a script that runs in the background (i.e., in another thread) without the page needing to wait for it to complete. The user can continue to interact with the page while the web worker runs in the background. Workers utilize thread-like message passing to achieve parallelism.
  1. **custom attributes in HTML5?**

A custom data attribute starts with data- and would be named based on your requirement. Following is the simple example−

<div class="example" data-subject="physics" data-level="complex">

...

</div>

The above will be perfectly valid HTML5 with two custom attributes called data-subject and data-level. You would be able to get the values of these attributes using JavaScript APIs or CSS in similar way as you get for standard attributes.

1. **Doc Type**

🡪 The <!DOCTYPE> declaration is not an HTML tag; it is an instruction to the web browser about what version of HTML the page is written in.

HTML5

<!DOCTYPE html>

1. **Different types Doctypes**

The <!**DOCTYPE**> declaration is not an **HTML** tag; it is an instruction to the web browser about what version of **HTML** the page is written in. In**HTML** 4.01, the <!**DOCTYPE**> declaration refers to a DTD, because**HTML** 4.01 was based on SGML.

1. **What is HTML Canvas?**

🡪 The HTML <canvas> element is used to draw graphics, on the fly, via JavaScript.

The <canvas> element is only a container for graphics. You must use JavaScript to actually draw the graphics.

Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

**Q. Difference between onload() and $.ready?**

The load event on the window and/or body element will fire once all the content of the page has been loaded -- this includes all images, scripts, etc... everything.

In contrast, jquery's $(document).ready(...) function will use a browser-specific mechanism to ensure that your handler is called as soon as possible after the HTML/XML dom is loaded and accessible. This is the earliest point in the page load process where you can safely run script that intends to access elements in the page's html dom. This point arrives earlier (often much earlier) than the final load event, because of the additional time required to load secondary resources (like images, and such).

We can have more than one document.ready() function in a page where we can have only one body onload function.

document.ready() function is called as soon as DOM is loaded where body.onload()function is called when everything gets loaded on the page that includes DOM, images and all associated resources of the page.

1. **What is difference between canvas and svg in html5?**

|  |  |  |
| --- | --- | --- |
|  | High Level Summary of Canvas vs. SVG  **Canvas**   1. Pixel based (Dynamic .png) 2. Single HTML element.(Inspect element in Developer tool. You can see only canvas tag) 3. Modified through script only 4. Event model/user interaction is granular (x,y) 5. Performance is better with smaller surface, a larger number of objects (>10k), or both   **SVG**   1. Shape based 2. Multiple graphical elements, which become part of the DOM 3. Modified through script and CSS 4. Event model/user interaction is abstracted (rect, path) 5. Performance is better with smaller number of objects (<10k), a larger surface, or both | |
| SVG is like a "draw" program. The drawing is specified as drawing instructions for each shape and any part of any shape can be changed. Drawings are shape-oriented.  Canvas is like a "paint" program. Once the pixels hit the screen, that is your drawing. You cannot change shapes except by overwriting them with other pixels. Paintings are pixel-oriented.  Being able to change drawings is very important for some programs; e.g. drafting apps, diagramming tools, etc. So SVG has an advantage here.  Being able to control individual pixels is important for some artistic programs.  Getting great animation performance for user-manipulation via mouse drags is easier with Canvas than SVG.  A single pixel on the computer screen will often consume 4 bytes of information and a computer screen these days takes several megabytes. So Canvas might be inconvenient if you want to let the user edit an image and then upload it again.  By contrast, drawing a handful of shapes that cover the entire screen using SVG takes up few bytes, downloads quickly, and can be uploaded again easily with the same advantages going in that direction as when it comes down on the other direction. So SVG can be faster than Canvas.  Google implemented Google Maps with SVG. That gives the web app its zippy performance and smooth scrolling. | |

There's a difference in what they are, and what they do for you.

* SVG is a document format for scalable vector graphics.
* Canvas is a javascript API for drawing vector graphics to a bitmap of a specific size.

To elaborate a bit, on format versus API:

With svg you can view, save and edit the file in many different tools. With canvas you just draw, and nothing is retained about what you just did apart from the resulting image on the screen. You can animate both, SVG handles the redrawing for you by just looking at the elements and attributes specified, while with canvas you have to redraw each frame yourself using the API. You can scale both, but SVG does it automatically, while with canvas again, you have to re-issue the drawing commands for the given size.

1. What is DTD in HTML?
2. what is new in HTML5?
3. How to overcome backword compatibility of HTML5 in IE?
4. What is HTML5 ?
5. What is WebSQL in HTML?

What is inline-block elements? Example of Default inline-block elements?

What is MVVM framework ?

how do you get HTML element properties ?

Can we have 2 head tag & does it through error?

* We can have 2 head tag but immediate to opening html tag give priority to show in output(e.g title)

################## --------- **CSS** -------- ###################

**Q. Box Model**

-🡪 div {

background-color: lightgrey;

width: 300px;

border: 25px solid green;

padding: 25px;

margin: 25px;

}

* 1. **media query**

Media Queries in Bootstrap allow you to move, show and hide content based on viewport size.

🡪 @media(min-width: 768px) {

.user-links {

a {

display: inline !important;

text-align: left;

line-height: normal;

&:nth-child(1) {

float: left;

}

&:nth-child(2) {

float: right;

}

}

}

.user-links-col1 {

top: -60px !important;

}

.user-links-col2 {

top: -59px !important;

}

}

1. **Selector**

🡪 .class, #id, \*(all element),

**ID is top priority in css to apply css on html tag(element)**

|  |  |  |
| --- | --- | --- |
|  | Element, element div, p | Selects all <div> elements and all <p> elements |

,

Element element div p Selects all <p> elements inside <div> elements,

|  |  |
| --- | --- |
| nth-child(2) p:nth-child(2) | Selects every <p> element that is the second child of its parent |

What is css specificity?

**Specificity** is a type of weighting that has a bearing on how your cascading style sheet (**CSS**) rules are displayed.

#divid ul li ?? or #divid .ulclass li ? which class will appear and why?

**################ ---- Bootstrap -------- #####################**

The Bootstrap grid system has four classes:

* xs (for phones)
* sm (for tablets)
* md (for desktops)
* lg (for larger desktops)

Some Bootstrap grid system rules:

* Rows must be placed within a .container (fixed-width) or .container-fluid (full-width) for proper alignment and padding
* Use rows to create horizontal groups of columns
* Content should be placed within columns, and only columns may be immediate children of rows
* Predefined classes like .row and .col-sm-4 are available for quickly making grid layouts
* Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and last column via negative margin on .rows
* Grid columns are created by specifying the number of 12 available columns you wish to span. For example, three equal columns would use three .col-sm-4

Offsets are a useful feature for more specialized layouts. They can be used to push columns over for more spacing, for example. The .col-xs = \* classes don't support offsets, but they are easily replicated by using an empty cell.

**Q. Diff betwn .container and .container-fluid**

.container-fluid has the CSS property width: 100%;, so it continually readjusts at every screen width granularity.

.container-fluid {

width: 100%;

}

.container has something like "width = 800px" (or em, rem etc.), a specific pixel width value at different screen widths. This of course is what causes the element width to abruptly jump to a different width when the screen width crosses a screen width threshold. And that threshold is governed by CSS3 media queries, which allow you to apply different styles for different conditions, such as screen width ranges.

@media screen and (max-width: 400px){

.container {

width: 123px;

}

}

@media screen and (min-width: 401px) and (max-width: 800px){

.container {

width: 456px;

}

}

@media screen and (min-width: 801px){

.container {

width: 789px;

}

}

**Q. Maintain Performance**

Refer to http://yslow.org/

1) Minimize HTTP Requests - Use CSS instead of images whenever possible, Combine multiple style sheets into one, Reduce scripts and put

them at the bottom of the page.

2) Optimize images using sprites, sprites allow you to reduce the number of HTTP requests,

which are one of the key points for fast web page loading.

2) Optimizing expensive CSS properties: Properties like box-shadow, gradient, border-radius, outline, opacity have

some interesting rendering behaviors that can slow down your UI.

3) Minify your JavaScript and CSS files

4) Load resources from a CDN

Minimize HTTP Requests - According to Yahoo, 80% of a Web page’s load time is spent downloading the different pieces-parts

of the page: images, stylesheets, scripts, Flash, etc. An HTTP request is made for each one of these elements, so the more

on-page components, the longer it takes for the page to render.

Use a Content Delivery Network

Avoid empty src or href

Put StyleSheets at the Top

Put Scripts at the Bottom

Avoid CSS Expressions

Make JavaScript and CSS External

Reduce DNS Lookups

Minify JavaScript and CSS

Avoid Redirects

Remove Duplicate Scripts

Make AJAX Cacheable

Use GET for AJAX Requests

Reduce the Number of DOM Elements

No 404s

HTML5 --- <!DOCTYPE html>

1. HTML vs XHTML difference
2. UL/LI elements -> display items in roman numbers
3. this in function / loop
4. how to create instance to a class
5. typeof var/func
6. reload in HTML5 without using javascript

   Ans: <meta http-equiv="refresh" content="0;URL='http://thetudors.example.com/'"/>

1. Jquery -> show/hide elements what is the method name (.show(), .hid())
2. ajax & post call in jquery -> $.push
3. Different ways of importing styles (internal,external,inline)
4. which one take more important in executing: inline
5. HTML5 features & new input tags
6. CSS3 features
7. how to design responsive designs without bootstrap

https://www.youtube.com/watch?v=eOG90Q8EfRo

1. how to handle file upload
2. Interface
3. abstract
4. override vs overloading
5. COR in jquery ajax - using : crossDomain: true,
6. Stop from submitting a form - using e.preventDefault();
7. Different ways of creating Arrays

Ans : var points = new Array();         // Bad  
 var points = [];                  // Good

1. html5 - svg tag

**Q. CSS Preprocessors?**

CSS Preprocesssors are like CSS on steroids. Preprocessors enhances the performance of regular style sheets by making more usuable and manageable.

SaaS,Less

Css counters

nthChildren in CSS

Css basics

Loadind css based on device resolution

**Q. How do you write a program to sum all the Array elements without using any loop?**

Ans) Using Recursive functions we can achieve

* + 1. **How do you load a script dynamically in JavaScript?**

Ans:

$(‘script’)

document.createElement(‘script’).src = “”;

Create dynamic element. document.body.append(scriptElement)

1. What problem does angular solve? Why is it introduced?
2. What is the difference when including the script within head and within the body ?
3. Can we load a css file in the body ? If not why?
4. In which tag do we load css files?

|  |
| --- |
|  |
| <link rel="stylesheet" href="[/css/site.css?cb=12](http://www.ng-newsletter.com/css/site.css?cb=12)"> |
|  |

1. How can you subclass/inherit from 2 different classes ? Write an example?

What is state and transitional in Doctype?

**Zen3 Questions:**

What is a Doctype?

What is background-size property in CSS3? And what is cover and plain?

Cover:

Scale the background image to be as large as possible so that the background area is completely covered by the background image.

How to find the length of all div elements in a html page?

var nodelist = document.body.childNodes.length;

What is the difference between ng-hover and mouseover?

[Expression](https://docs.angularjs.org/guide/expression) to evaluate upon mouseover. ([Event object is available as $event](https://docs.angularjs.org/guide/expression#-event-))

How do you display a sidebar display and collapse?

How do you set multiple background images ? In IE it doesnt work so how do you make sure it works in IE?

background: {

url(number.png) 600px 10px no-repeat, /\* On top, like z-index: 4; \*/

url(thingy.png) 10px 10px no-repeat, /\* like z-index: 3; \*/

url(Paper-4.png); /\* On bottom, like z-index: 1; \*/

}

**Difference between block, inline and inline-block?**

(http://dustwell.com/div-span-inline-block.html)

An inline element has no line break before or after it, and it tolerates HTML elements next to it.

A block element has some whitespace above and below it and does not tolerate any HTML elements next to it.

An inline-block element is placed as an inline element (on the same line as adjacent content), but it behaves as a block element.

An element with display: inline; cannot have a height or a width or a vertical margin.

An element with display: block; can have a width, height and margin.

If you want to add a width/height/margins to the inline element, you need to set the element to display: inline-block;

**Q. What is polyfill in html5?**

In web development, a polyfill is a downloadable piece of code which provides facilities

that are not built into a web browser. It implements technology that a developer expects the browser

to provide natively, providing a more uniform API landscape. For ex the functionality you expect to work in

modern browsers to work in older browsers. Ie to support canvas (an html5 feature) in older browsers.

in that function. So, if a variable is declared in the middle or even at the very bottom of a function, that variable will

still be visible even at the top of the function – meaning that the variable will be visible above its declaration. But, this

does not mean that assigned values will still remain associated with the variable – it’s just that the variable name will be

recognized starting from the very beginning of the function.This is an interesting feature of Javascript known as hoisting,

because it is as if Javascript is “hoisting” (which means to move up) the variable declaration to the top of the function.

**Q. Can I add multiple footer elements in HTML5 ?**

**Yes**

**Q.** **Does browser understands less ? How does less file is converted to css (how or who converts it)?**

**Q. How do you check or add a condition if scroll bar is at the bottom of the container ?**

Ans) element.scrollHeight - element.scrollTop === element.clientHeight

**Q. What is less and how does it work ?**

Less is a CSS pre-processor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions and many other techniques that allow you to make CSS that is more maintainable, themable and extendable.

Less runs inside Node, in the browser and inside Rhino. There are also many 3rd party tools that allow you to compile your files and watch for changes.

**Q. How do you measure performance of your application and how can you optmize it ?**

**Q.** **How to vertical align a text or container in another div container ?**

Ans)

<div style="height: 400px;display: table-cell; vertical-align: middle;">

         <span> everything is vertically centered in modern IE8+ and others.</span>

</div>

What is CDN?

What is Load Balancer?

What is Grunt and Bower?

http://www.pdfescape.com/open/?BEC0D3AA33EE8272E2BE834116217E1F596B8AF6D6825FAB

**XMTP, SOAP, REST Protocol**

[**https://www.youtube.com/watch?v=FoHxr0R9D7U**](https://www.youtube.com/watch?v=FoHxr0R9D7U)

The decision between the two will be your first choice in designing a web service, so it is important to understand the pros and cons of the two. It is also important, in the sometimes heated debate between the two philosophies, to separate reality from rhetoric.

**REST fundamentals**

* Everything in REST is considered as a resource.
* Every resource is identified by an URI.
* Uses uniform interfaces. Resources are handled uing POST, GET, PUT, DELETE operations which are similar to Create, Read, update and Delete(CRUD) operations.
* Be stateless. Every request is an independent request. Each request from client to server must contain all the information necessary to understand the request.
* Communications are done via representations. E.g. XML, JSON RESTful Web Services A RESTFul web services are based on HTTP methods and the concept of REST. A RESTFul web service typically defines the base URI for the services, the supported MIME-types (XML, text, JSON, user-defined, ...) and the set of operations (POST, GET, PUT, DELETE) which are supported.

### Fundamental REST Principles

**Client-Server Communication**

Client-server architectures have a very distinct separation of concerns. All applications built in the RESTful style must also be client-server in principle.

**Stateless**

Each client request to the server requires that its state be fully represented. The server must be able to completely understand the client request without using any server context or server session state. It follows that all state must be kept on the client.

**Cacheable**

Cache constraints may be used, thus enabling response data to be marked as cacheable or not-cacheable. Any data marked as cacheable may be reused as the response to the same subsequent request.

**Uniform Interface**

All components must interact through a single uniform interface. Because all component interaction occurs via this interface, interaction with different services is very simple. The interface is the same! This also means that implementation changes can be made in isolation. Such changes, will not affect fundamental component interaction because the uniform interface is always unchanged. One disadvantage is that you are stuck with the interface. If an optimization could be provided to a specific service by changing the interface, you are out of luck as REST prohibits this. On the bright side, however, REST is optimized for the web, hence incredible popularity of REST over HTTP!

The above concepts represent defining characteristics of REST and differentiate the REST architecture from other architectures like web services. It is useful to note that a REST service is a web service, but a web service is not necessarily a REST service.

**SOAP fundamentals**

* WSDL defines contract between client and service and is static by its nature.
* SOAP builds an XML based protocol on top of HTTP or sometimes TCP/IP.
* SOAP describes functions, and types of data.
* SOAP is a successor of XML-RPC and is very similar, but describes a standard way to communicate.
* Several programming languages have native support for SOAP, you typically feed it a web service URL and you can call its web service functions without the need of specific code.
* Binary data that is sent must be encoded first into a format such as base64 encoded.
* Has several protocols and technologies relating to it: WSDL, XSDs, SOAP, WS-Addressing.

**Q.** **explian agile ?**

Agile development model is a type of Incremental model. Software is developed in incremental, rapid cycles from the start of the project,

instead of trying to deliver it all at once near the end.

**Q.** **Explain Scrum?**

Scrum is an iterative and incremental agile software development methodology for managing product development.

We follow Scrum methodology which involves Sprint planning, Daily Scrum, Sprint review meeting and Sprint retrospective meeting.