**Advanced JavaScript:**

**Event Bubbling and Capturing**

Event bubbling and capturing are two ways of event propagation in HTML DOM.

* Event bubbling
* In bubbling the event is first captured and handled by the inner most elements and then propagated to outer elements.

**Ex:**

<div class="div-1">1

<div class="div-2">2

<div class="div-3">3</div>

</div>

</div>

* Event Capturing
* In capturing the event is first captured by the outer most elements and propagated to the inner most elements.

Only event bubbling model is supported by all the major browsers.

So if you are going to use event capturing still you need to handle event bubbling for IE. So it will easier to use event bubbling instead of capturing.

**Callbacks and Anonymous functions**

**Callbacks**

Callback functions are the functions that are passed to another function as an argument.

Callback functions are probably the most widely used functional programming technique in JavaScript, and they are literally in just about every piece of JavaScript and Jquery code.

Ex1:

function getDivision(arg1, arg2, callback) {

var divisionValue = arg1/arg2;

callback(divisionValue);

}

getDivision(5, 15, function(num){

return Math.round(num);

});

Ex2:

$("button").click (function () {  
 $("p").hide (1000);  
  alert ("The paragraph is now hidden");  
});

Ex3:

$("button").click(function(){  
  $("p").hide("slow”, function(){  
    alert("The paragraph is now hidden");  
  });  
});

**Anonymous function**

<http://helephant.com/2008/08/23/javascript-anonymous-functions/>

Anonymous functions are functions that are dynamically declared at runtime. They’re called anonymous functions because they aren’t given a name in the same way as normal functions.

**Normal function declaration**

**function** hello() {

alert('world');

}

hello();

**Anonymous function declaration**

**var** anon = **function**() {

alert('I am anonymous');

};

anon();

setTimeout(**function**() {

alert('hello');

}, 1000);

(**function**() {

alert('foo');

}());

**Closures:**

**http://javascriptissexy.com/understand-javascript-closures-with-ease/**

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.

The inner function has access not only to the outer function’s variables, but also to the outer function’s parameters. Note that the inner function cannot call the outer function’s *arguments* object, however, even though it can call the outer function’s parameters directly.

EX1:

function showName (firstName, lastName) {

var nameIntro = "Your name is";

function makeFullName() {

return nameIntro + firstName + " " + lastName;

}

return makeFullName();

}

showName("Michael", "Jackson");

Ex2:

function celebrityName(firstName) {

var nameIntro = "This celebrity is";

function lastName(theLastName) {

return nameIntro + firstName + " " + theLastName;

}

return lastName;

}

var mjName = celebrityName("Michael");

mjName("Jackson");

**OOPS in JavaScript**

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction\_to\_Object-Oriented\_JavaScript

Object-oriented programming is a programming paradigm that uses abstraction to create models based on the real world.

It uses several techniques from previously established paradigms, including modularity, polymorphism, and encapsulation.

Object-oriented programming is intended to promote greater flexibility and maintainability in programming, and is widely popular in large-scale software engineering.

**Concepts:**

* Class
* Object
* Property
* Method
* Constructor
* Inheritance
* Encapsulation
* Abstraction
* Polymorphism