**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

Class: JavaScript is a prototype-based language which contains no class statement, such as is found in C++ or Java.

Ex:

function Person() { }

Object:

An object is just a special kind of data, with **properties** and **methods**.

Ex:

function Person() { }

var person1 = new Person();

var person2 = new Person();

Property:

Properties are variables contained in the class; every instance of the object has those properties. Properties should be set in the prototype property of the class (function) so that inheritance works correctly.

Ex:

function Person(gender) {

this.gender = gender;

alert('Person instantiated');

}

var person1 = new Person('Male');

var person2 = new Person('Female');

alert('person1 is a ' + person1.gender);

Method:

Methods follow the same logic as properties; the difference is that they are functions and they are defined as functions. Calling a method is similar to accessing a property, but you add () at the end of the method name, possibly with arguments.

Ex:

function Person(gender) {

this.gender = gender;

this.sayHello = function()

{

alert ('hello');

};

}

var person1 = new Person('Male');

var person2 = new Person('Female');

person1.sayHello();

Constructor:

The constructor is called at the moment of instantiation (the moment when the object instance is created). The constructor is a method of the class. In JavaScript, the function serves as the constructor of the object; therefore, there is no need to explicitly define a constructor method. Every action declared in the class gets executed at the time of instantiation.

Ex:

function Person() {

alert('Person instantiated');

}

var person1 = new Person();

var person2 = new Person();

Inheritance:

Inheritance is a way to create a class as a specialized version of one or more classes (*JavaScript only supports single class inheritance*).

Ex:

// define the Person Class

function Person() {}

Person.prototype.walk = function(){

alert ('I am walking!');

};

Person.prototype.sayHello = function(){

alert ('hello');

};

// define the Student class

function Student() {

// Call the parent constructor

Person.call(this);

}

// inherit Person

Student.prototype = new Person();

// correct the constructor pointer because it points to Person

Student.prototype.constructor = Student;

// replace the sayHello method

Student.prototype.sayHello = function(){

alert('hi, I am a student');

}

// add sayGoodBye method

Student.prototype.sayGoodBye = function(){

alert('goodBye');

}

var student1 = new Student();

student1.sayHello();

student1.walk();

student1.sayGoodBye();

// check inheritance

alert(student1 instanceof Person); // true

alert(student1 instanceof Student); // true

Encapsulation:

In the previous example, Student does not need to know how the Person class's walk() method is implemented, but still can use that method; the Student class doesn't need to explicitly define that method unless we want to change it. This is called encapsulation, by which every class inherits the methods of its parent and only needs to define things it wishes to change.

Abstraction:

Abstraction is a mechanism that permits modeling the current part of the working problem. This can be achieved by inheritance (specialization), or composition. JavaScript achieves specialization by inheritance, and composition by letting instances of classes be the values of attributes of other objects.

Polymorphism:

Just like all methods and properties are defined inside the prototype property, different classes can define methods with the same name; methods are scoped to the class in which they're defined. This is only true when the two classes do not hold a parent-child relation (when one does not inherit from the other in a chain of inheritance).

Prototype:

http://net.tutsplus.com/tutorials/javascript-ajax/prototypes-in-javascript-what-you-need-to-know/