Quotes

“People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on each other, like a wall of mini stones” - Donald Knuth

Notes:

1. All functions in JavaScript have properties, just as objects have properties.
2. All functions are considered objects in JavaScript.
3. In a web page, global variables belong to the window object.
4. Variables created without the keyword var, are always global, even if they are created inside a function.
5. A function defined as the property of an object, is called a method to the object.
6. A function designed to create new objects, is called an object constructor.
7. JavaScript has function level scoping.
8. All variables and functions defined within the anonymous function aren’t available to the code outside of it, effectively using closure to seal itself from the outside world.
9. A function is just a special kind of object, and like any object a function can have properties.
10. Functions automatically get a property called prototype, which is just an empty object.
11. Function is a tool to structure larger programs, to reduce repetition, to associate names with subprograms, and to isolate these subprograms from each other.

Return Statement

* A return statement determines the value the function returns. When control comes across such a statement, it immediately jumps out of the current function and gives the returned value to the code that called the function. The return keyword without an expression after it will cause the function to return undefined.

Keep in Mind

* Languages such as Java provide the ability to declare methods private, meaning that they can only be called by other methods in the same class.
* Private methods aren't just useful for restricting access to code: they also provide a powerful way of managing your global namespace, keeping non-essential methods from cluttering up the public interface to your code.

JavaScript Name Validator

* Search in google like JavaScript variable name validator or link in below
* <https://mothereff.in/js-variables>

1. JavaScript function is a series of statement that group together in a package

Objects

* Objects is nothing one data type more than many data types
* Objects are flexible they are hold many other data types like strings, array, functions, other objects and so many.
* Objects always start and end with curly braces
* Inside their we have number of flexible properties
* And separate by name and value pairs
* Example of object look like

var info = {

full\_name : “Ray Villalobos”,

title: “Staff Author”,

links: [

{ blog : “<http://iviewsource.com>”},

{ twitter : “http://twitter.com/planetoftheweb”}

]

};

Function Basics

* All functions defined with keyword function
* Functions have to be declared before use
* In JS function name sometimes optional
* Function parameters – are only available inside function
* Invoking a function – function calls automatically

Functions Declaration

Traditional approach

function plus(a,b){

return a+b;

}

plus(2,4);

Definition Expressions

var plus = function(a,b) {

return console.log(a+b);

};

plus(2,2);

* Assigning functions to expressions
* After a function expression has been stored in a variable, the variable can be used as a function
* The function above is actually an anonymous function (a function without a name).
* Functions stored in variables do not need function names. They are always invoked (called) using the variable name.
* The function above ends with a semicolon because it is a part of an executable statement.

Anonymous functions

* More flexible than expressions
* Can be invoked immediately
* Can initialize values immediately
* Useful when needed once or execute something immediately
* Function expressions will execute automatically if the expression is followed by ().
* You have to add parentheses around the function to indicate that it is a function expression.

Self-Invoking Anonymous Function

* A self-invoking anonymous runs automatically/immediately when you create it and has no name, hence called anonymous.
* Example

(function(){

// some code…

})();

* Another Example

var anon = function() {

alert('I am anonymous');

};

anon();

* The most common use for anonymous functions are as arguments to other functions, or as a closure.
* Example

setTimeout(function() {

alert('hello');

}, 1000);

// Our anonymous function is passed to setTimeout, which will execute

// the function in 1000 milliseconds.

(function() {

alert('foo');

})();

// This is a common method of using an anonymous function as a closure,

// which many JavaScript frameworks use.

Breakdown of the above anonymous statements:

* The surrounding braces is a wrapper for the anonymous function
* The trailing braces initiates a call to the function and can contain arguments

(function(message) {

alert(message);

}('foo'));

// Another way to write the previous example and get the same result

* Calculating factorial by anonymous function

// returns the factorial of 10.

alert((function factorial(n) {

return (n <= 1)

? 1

: factorial(n - 1) \* n;

})(10));

Method

* Method is nothing more than function
* A function is called method if its assigned to a property of an object

Function Invocation

* Function will not execute until call or invoked.
* Four ways of invoking functions
* Functions
* Methods
* Constructors
* Call & Apply methods
* First two are most common
* Receive arguments & this
* Arguments parameters are useful in the case of multiple parameters
* Invoking stops current execution
* plus(a,b)
* Traditional invocation
* this parameter bound to global object
* Using functions as objects
* Objects are flexible because they hold any other data types like variables or methods etc.
* Object start with curly braces
* this attributes gets the value of the object which is really useful.

JavaScript: Functions Invocation

Invoking Functions as methods

* The this argument points to the object
* Invoke the function using dot notation
* The binding of this happens at invocation(call) time
* So the this attribute can’t be bound to the object until we invoke the method

Invoking through the constructor

* Functions can construct objects
* Using the new keyword
* new creates a new instance of the object (it’s like copy of object not like that it’s like reproduction)
* Each new instance has its own set of properties
* this argument points to the instance of the object
* Constructor names are capitalized
* This little bit gives the instructor to us or other developers that this is special like constructor.
* If we call this within the object then this will referring the object otherwise this will refereeing the global object.
* In general, Constructor is also behaves and acts as function.

Expanding objects through prototype

* Prototypal Inheritance
* JavaScript known as prototypal inheritance language
* Every object can be based on another object
* We don’t build same functionality for different things whether we do this linking an object by prototype object to another.
* prototype object give us access
* Multiple objects can inherit same functionality
* All objects inherit properties
* Declarations inherit from Function
* Function constructor inherits from Object
* With the prototype object we can expand the functionality of anything in JavaScript.

Invoking through Call & Apply

* Sometimes call & apply techniques known as
* Indirect invocation
* Both this function give little bit better control over this argument
* One of the biggest problem in traditional function decoration that this parameter is bound to the global object.
* Define the value of this argument
* Generally this refers to global object
* Call and apply take two parameters
* Control: this and arguments
* Call passes a value, Apply passes an array
* One of the drawbacks of traditional function invocation is this parameter is bound to the global object.

The difference between Call & Apply

* The difference is that apply lets you invoke the function with arguments as an array; call requires the parameters be listed explicitly.
* A useful mnemonic is "A for array and C for comma."
* Pseudo syntax:

theFunction.apply(valueForThis, arrayOfArgs)

theFunction.call(valueForThis, arg1, arg2, ...)

* Sample code:

function theFunction(name, profession) {

alert("My name is " + name + " and I am a " + profession + ".");

}

theFunction("John", "fireman");

theFunction.apply(undefined, ["Susan", "school teacher"]);

theFunction.call(undefined, "Claude", "mathematician");

This

* This keyword as a shortcut, a referent; it refers to an object; that is, the subject in context, or the subject of the executing code.
* In General Scenario we write as like

John is running fast because he is trying to catch the train.

* Note the use of the pronoun “he.” We could have written this: “John is running fast because John is trying to catch the train.” We don’t reuse “John” in this manner, for if we do, our family, friends, and colleagues would abandon us.
* The this keyword not only refers to the object but it also contains the value of the object.
* The this reference ALWAYS refers to (and holds the value of) an object—a singular object—and it is usually used inside a function or a method, although it can be used outside a function in the global scope.
* this is not assigned a value until an object invokes the function where this is defined.

In Concise this keyword is

* The thing called this, is the object that "owns" the current code.
* The value of this, when used in a function, is the object that "owns" the function.

The use of this in the global scope

* In the global scope, when the code is executing in the browser, all global variables and functions are defined on the window object.

The arguments parameter

* List of elements passed
* An array like object
* We call the arguments with an index
* Numerical index arguments[x]
* We can get the arguments.length
* We can loop through arguments
* Can’t use all array methods like pop, push, shift or others.
* This type of object is useful if we pass unknown number of arguments.

The return statement

* When we invoke a function it generates like an equation.
* An equation has usually result
* Returns an expression
* Sort of optional
* Only in function body
* Return sends something back to the caller
* Return statement are usually last step of the function
* Stop execution of the function
* You can have more than one
* So we can use if statement and return a value and stop the execution of the function.
* The return statement can be return anything or nothing at all
* Careful with auto semicolon insertion
* JavaScript has wired feature make semicolon optional.

Var plus = function(a,b){

Return

A+b

}  
console.log(plus(2,2));

* Here JavaScript interpret return as own statement and execute the function.

Anonymous Closures

* JavaScript variables can belong to the local or global scope.
* Private variables can be made possible with closures.
* Closure is nothing but a data structure storing a function together with an environment, which maps all associating each free variable of the function (that are used locally, but defined in an enclosing scope) with the value or storage location the name was bound to at the time the closure was created. A closure allows the function to access those captured variables through the closure's reference to them, even when the function is invoked outside its scope.
* 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.
* A closure is a function having access to the parent scope, even after the parent function has closed.
* A closure is a special kind of object that combines two things: a function, and the environment in which that function was created.
* A closure lets you associate some data (the environment) with a function that operates on that data.
* This has obvious parallels to object oriented programming, where objects allow us to associate some data (the object's properties) with one or more methods.
* Consequently, you can use a closure anywhere that you might normally use an object with only a single method.
* Much of the code we write in web JavaScript is event-based — we define some behavior, then attach it to an event that is triggered by the user (such as a click or a key-press).
* Our code is generally attached as a callback: a single function which is executed in response to the event.

Emulating private methods with closures

* Languages such as Java provide the ability to declare methods private, meaning that they can only be called by other methods in the same class.
* JavaScript does not provide a native way of doing this, but it is possible to emulate private methods using closures. Private methods aren't just useful for restricting access to code: they also provide a powerful way of managing your global namespace, keeping non-essential methods from cluttering up the public interface to your code.

Performance considerations

* It is unwise to unnecessarily create functions within other functions if closures are not needed for a particular task, as it will negatively affect script performance both in terms of processing speed and memory consumption.
* For instance, when creating a new object/class, methods should normally be associated to the object's prototype rather than defined into the object constructor. The reason is that whenever the constructor is called, the methods would get reassigned (that is, for every object creation).

Variable Scope and Hoisting

In most Programming languages available scopes are

* Scope: life and death of a variable
* Block Scope: Variables live within{}

JavaScript has special kinds of scope like

* Function scope: variables live in functions
* If a variable create in parent functions it also available in child functions

Scope Chain

* A variable find its variables within the functions
* It also looks up through all the parent functions

Global Variables

* Dangerous global variables because global variables are available in the whole web application
* Global variables may cause lot of functions may be used a variable with the same name.
* Always used the var when declare a variable.

Dangerous things in JS

* Variable definitions are hoisted

JavaScript Hoisting

* Hoisting is JavaScript's default behavior of moving declarations to the top.
* 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).
* Hoisting is (to many developers) an unknown or overlooked behavior of JavaScript.
* If a developer doesn't understand hoisting, programs may contain bugs (errors).
* To avoid bugs, always declare all variables at the beginning of every scope.
* Since this is how JavaScript interprets the code, it is always a good rule.

Creating and namespacing modules

* Modules let us reuse code across different applications
* Modules as like a library used across different applications
* When we create a module then we first create a namespace for it
* Namespacing protects variables
* Namespacing allow us to protect any variables in our module from any global scoped variables
* Return statement communicates back
* To the rest of the applications
* Instead of returning a statement we return an object

Short-circuit evaluation

* Short-circuit evaluation, minimal evaluation, or McCarthy evaluation denotes the semantics of some Boolean operators in some programming languages in which the second argument is executed or evaluated only if the first argument does not suffice to determine the value of the expression: when the first argument of the AND function evaluates to false, the overall value must be false; and when the first argument of the OR function evaluates to true, the overall value must be true.

Short Circuit Operators in JavaScript

* The && and || operators are called short-circuit operators. They will return the value of the second operand based on the value of the first operand.
* The && operator is useful for checking for null objects before accessing their attributes. For example...
* var name = person && person.getName();
* This code is the same as

if(person) {

var name = person.getName();

}

* The || operator is used for setting default values.
* var name = persons\_name || "John Doe";
* The equivalent code is

if(persons\_name) {

var name = persons\_name;

} else {

var name = "John Doe";

}

Chaining module method

* It’s simple trick that allows one function to call another

So in a Nutshell

* JavaScript functions can be invoked in 4 different ways.
* The code in a function is not executed when the function is defined. It is executed when the function is invoked.
* Some people use the term "call a function" instead of "invoke a function".
* JavaScript function can be invoked without being called.
* Invoking a Function as a Function

Example:

function myFunction(a, b) {

return a \* b;

}

myFunction(10, 2); // myFunction(10, 2) will return 20

The function above does not belong to any object. But in JavaScript there is always a default global object.

In HTML the default global object is the HTML page itself, so the function above "belongs" to the HTML page.

In a browser the page object is the browser window. The function above automatically becomes a window function.

myFunction() and window.myFunction() is the same function

* Invoking a Function as a Method

E.g.

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.

E.g.

// 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"

- A constructor invocation creates a new object. The new object inherits the properties and methods from its constructor.

- The value of this will be the new object created when the function is invoked.

- 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

Example call method

function myFunction(a, b) {

return a \* b;

}

myObject = myFunction.call(myObject, 10, 2); // Will return 20

Example apply method

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.
* 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.