Objetos

Contienen propiedades y las propiedades nombre y valor.

Pueden contener otros objetos dentro.

Son "CLASES GRATIS" class-free en JS

```
var object={};
      var store = {
      "fist-name" : "Josue",
      "last-name" : "Solis"
      };
      var flight = {
          airline: "Oceanic",
          number: 815,
          departure: {
              IATA: "SYD",
              time: "2004-09-22 14:55",
              city: "Sydney"
          },
          arrival: {
              IATA: "LAX",
              time: "2004-09-23 10:42",
              city: "Los Angeles"
          }
      };
```

Llamado

```
flight.departure.IATA // "SYD"
```

Editar

```
stooge['first-name'] = 'Jerome';
```

Referencia

```
var x = stooge;
x.nickname = 'Curly';
var nick = stooge.nickname;
    // nick is 'Curly' because x and stooge
     // are references to the same object
```

Prototype

Every object is linked to a prototype object from which it can inherit properties. All objects created from object literals are linked to Object.prototype , an object that comes standard with JavaScript.

Ejemplo Prototype

```
if (typeof Object.create !== 'function') {
     Object.create = function (o) {
         var F = function () {};
         F.prototype = o;
         return new F();
     };
    }
var another_stooge = Object.create(stooge);
```

Reflection

It is easy to inspect an object to determine what properties it has by attempting to retrieve the properties and examining the values obtained. The typeof operator can be very helpful in determining the type of a property:

```
typeof flight.number      // 'number'
typeof flight.status      // 'string'
typeof flight.arrival     // 'object'
typeof flight.manifest    // 'undefined'
```

Enumeration

```
var i;
var properties = [
    'first-name',
    'middle-name',
    'last-name',
    'profession'
];
for (i = 0; i < properties.length; i += 1) {
    document.writeln(properties[i] + ': ' +
    another_stooge[properties[i]]);
}
```

Delete

```
delete another_stooge.nickname;
```

Global Abatement

```
var MYAPP = {};
MYAPP.stooge = {
    "first-name": "Joe",
    "last-name": "Howard"
};
MYAPP.flight = {
    airline: "Oceanic",
    number: 815,
    departure: {
        IATA: "SYD",
        time: "2004-09-22 14:55",
        city: "Sydney"
    },
    arrival: {
        IATA: "LAX",
        time: "2004-09-23 10:42",
        city: "Los Angeles"
    }
};
```

• Funciones

The best thing about JavaScript is its implementation of functions. A function encloses a set of statements. Functions are the fundamental modular unit of JavaScript.

Functions Objects

```
Functions in JavaScript are objects.
```

Function Literal

A function literal has four parts:

1. Reserved word function
2. Function name
3. Set of parameters
4. Curly braces {}

```
var add = function (a, b) {
    return a + b;
};
```

Invocation

Invoking a function suspends the execution of the current function, passing control and parameters to the new function.

The method invocation pattern

When a function is stored as a property of an object, we call it a method.

```
var myObject = {
    value: 0,
    increment: function (inc) {
        this.value += typeof inc === 'number' ? inc : 1;
    }
};
myObject.increment();
document.writeln(myObject.value);    // 1
myObject.increment(2);
document.writeln(myObject.value);    // 3
```

The function invocation pattern

When a function is not the property of an object, then it is invoked as a function.

```
var sum = add(3, 4);    // sum is 7

// Augment myObject with a double method.
myObject.double = function () {
    var that = this;    // Workaround.
    var helper = function () {
        that.value = add(that.value, that.value);
    };
    helper();    // Invoke helper as a function.
};

// Invoke double as a method.
myObject.double();
document.writeln(myObject.getValue());    // 6
```

The constructor invocation pattern

JavaScript is a prototypal inheritance language. That means that objects can inherit properties directly from other objects. The language is class-free.
When a function is not the property of an object, then it is invoked as a function.

```
// Create a constructor function called Quo.
// It makes an object with a status property.
var Quo = function (string) {
    this.status = string;
};

// Give all instances of Quo a public method
// called get_status.
Quo.prototype.get_status = function () {
    return this.status;
};

// Make an instance of Quo.
var myQuo = new Quo("confused");
document.writeln(myQuo.get_status());  // confused
```

Patrones de Invocación

Arguments

When they are invoked is the arguments array. It gives the function access to all of the arguments that were sup plied with the invocation, including excess arguments that were not assigned to parameters.

```
var sum = function () {
    var i, sum = 0;
    for (i = 0; i < arguments.length; i += 1) {
        sum += arguments[i];
    }
    return sum;
};
document.writeln(sum(4, 8, 15, 16, 23, 42)); // 108
```

Return

The return statement can be used to cause the function to return early. When return is executed, the function returns immediately without executing the remaining statements. A function always returns a value. If the return value is not specified, then undefined is returned.

```
var sum = function () {
    var i, sum = 0;
    for (i = 0; i < arguments.length; i += 1) {
        sum += arguments[i];
    }
    return sum;
};
document.writeln(sum(4, 8, 15, 16, 23, 42)); // 108
```

Exceptions

JavaScript provides an exception handling mechanism. Exceptions are unusual (but not completely unexpected) mishaps that interfere with the normal flow of a program. When such a mishap is detected, your program should throw an exception:

```
var add = function (a, b) {
     if (typeof a !== 'number' || typeof b !== 'number') {
          throw {
               name: 'TypeError',
               message: 'add needs numbers'
          };
     }
     return a + b;
```

```
}
```

The throw statement interrupts execution of the function.
The exception object will be delivered to the catch clause of a try statement:

```
// Make a try_it function that calls the new add
// function incorrectly.
var try_it = function ( ) {
      try {
            add("seven");
      } catch (e) {
            document.writeln(e.name + ': ' + e.message);
      }
}
try_it( );
```

Augmenting Types

JavaScript allows the basic types of the language to be augmented.
Object.prototype makes that method available to all objects.

```
// Add a method conditionally.
Function.prototype.method = function (name, func) {
      if (!this.prototype[name]) {
            this.prototype[name] = func;
      }
      return this;
};
```

Recursion

A recursive function is a function that calls itself, either directly or indirectly.
The Towers of Hanoi Puzzle.

```
var hanoi = function (disc, src, aux, dst) {
      if (disc > 0) {
            hanoi(disc - 1, src, dst, aux);
            document.writeln('Move disc ' + disc +
                              ' from ' + src + ' to ' + dst);
            hanoi(disc - 1, aux, src, dst);
      }
};
hanoi(3, 'Src', 'Aux', 'Dst');
```