Objetos de JavaScript

Ya has aprendido que las variables de JavaScript son contenedores para valores de datos.

Este código asigna un valor simple (Fiat) a una variable llamada automóvil:

```
var car = "Fiat";
```

Inténtalo tú mismo "

Los objetos también son variables. Pero los objetos pueden contener muchos valores.

Este código asigna muchos valores (Fiat, 500, blanco) a una variable llamada automóvil:

```
var car = {type:"Fiat", model:"500", color:"white"};
```

Inténtalo tú mismo "

Los valores se escriben como nombre: pares de valores (nombre y valor separados por dos puntos).

Los objetos JavaScript son contenedores para valores con nombre .

Propiedades del objeto

El nombre: pares de valores (en objetos JavaScript) se llaman propiedades .

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

Propiedad	El valor de la propiedad
nombre de pila	John
apellido	Gama
años	50
color de los ojos	azul

Métodos de objeto

Los métodos son acciones que se pueden realizar en los objetos.

Los métodos se almacenan en propiedades como definiciones de funciones .

Propiedad	El valor de la propiedad
nombre de pila	John
apellido	Gama
años	50
color de los ojos	azul
nombre completo	function () {return this.firstName + "" + this.lastName;}

Los objetos JavaScript son contenedores para valores denominados denominados propiedades o métodos.

Definición de objeto

Usted define (y crea) un objeto JavaScript con un objeto literal:

Ejemplo

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

Inténtalo tú mismo "

Los espacios y los saltos de línea no son importantes. Una definición de objeto puede abarcar varias líneas:

Ejemplo

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

Inténtalo tú mismo "

Acceso a las propiedades del objeto

Puede acceder a las propiedades del objeto de dos maneras:

```
objectName.propertyName
```

o

```
objectName["propertyName"]
```

Ejemplo 1

```
person.lastName;
```

Inténtalo tú mismo "

Ejemplo2

```
person["lastName"];
```

Inténtalo tú mismo "

Acceso a métodos de objetos

Accede a un método de objeto con la siguiente sintaxis:

```
objectName.methodName()
```

Ejemplo

```
name = person.fullName();
```

Inténtalo tú mismo "

Si accede al método fullName , sin (), devolverá la definición de función :

Ejemplo

```
name = person.fullName;
```

Inténtalo tú mismo "

Un método es en realidad una definición de función almacenada como un valor de propiedad.

¡No declarar cadenas, números y booleanos como objetos!
Cuando se declara una variable de JavaScript con la palabra clave "nuevo", la variable se crea como un objeto:

```
var   x = new String();       // Declares x as a String object
var y = new Number();       // Declares y as a Number object
var z = new Boolean();      // Declares z as a Boolean object
```

Evita String, Number y objetos booleanos. Complican su código y ralentizan la velocidad de ejecución.

Aprenderá más sobre los objetos más adelante en este tutorial.

¡Pruébelo con ejercicios!

Ejercicio 1 » Ejercicio 2» Ejercicio 3 »

Objetos de JavaScript

En JavaScript, los objetos son el rey. Si entiendes objetos, entiendes JavaScript.

En JavaScript, casi "todo" es un objeto.

- Los booleanos pueden ser objetos (si se definen con la nueva palabra clave)
- Los números pueden ser objetos (si se definen con la nueva palabra clave)
- Las cadenas pueden ser objetos (si se definen con la nueva palabra clave)
- Las fechas son siempre objetos
- Las matemáticas son siempre objetos.
- Las expresiones regulares son siempre objetos
- Las matrices son siempre objetos.
- Las funciones son siempre objetos.
- Los objetos siempre son objetos

Todos los valores de JavaScript, excepto primitivas, son objetos.

Primitivas de JavaScript

Un valor primitivo es un valor que no tiene propiedades ni métodos.

Un tipo de datos primitivo es un dato que tiene un valor primitivo.

JavaScript define 5 tipos de tipos de datos primitivos:

- string
- number
- boolean
- null
- undefined

Los valores primitivos son inmutables (están codificados y por lo tanto no se pueden cambiar).

si x = 3.14, puede cambiar el valor de x. Pero no puede cambiar el valor de 3.14.

Value	Type	Comment
"Hello"	string	"Hello" is always "Hello"
3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

Usando un Objeto Literal

Esta es la forma más fácil de crear un objeto JavaScript.

Usando un objeto literal, ambos definen y crean un objeto en una instrucción.

Un objeto literal es una lista de nombres: pares de valores (como la edad: 50) dentro de llaves {}.

El siguiente ejemplo crea un nuevo objeto JavaScript con cuatro propiedades:

Ejemplo

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

Inténtalo tú mismo "

Los espacios y los saltos de línea no son importantes. Una definición de objeto puede abarcar varias líneas:

Ejemplo

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

Inténtalo tú mismo "

Uso de la palabra clave JavaScript new

El siguiente ejemplo también crea un nuevo objeto JavaScript con cuatro propiedades:

Ejemplo

```
var   person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Inténtalo tú mismo "

Los dos ejemplos anteriores hacen exactamente lo mismo. No es necesario usar un nuevo objeto ().
Por simplicidad, legibilidad y velocidad de ejecución, use el primero (el método literal del objeto).

Usar un constructor de objetos

Los ejemplos anteriores son limitados en muchas situaciones. Solo crean un solo objeto.

A veces nos gusta tener un "tipo de objeto" que se puede usar para crear muchos objetos de un tipo.

La forma estándar de crear un "tipo de objeto" es utilizar una función de constructor de objetos:

Ejemplo

```
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");
```

Inténtalo tú mismo "

La función anterior (persona) es un constructor de objetos.
Una vez que tenga un constructor de objetos, puede crear nuevos objetos del mismo tipo:

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

La palabra clave this

En JavaScript, la cosa llamada this es el objeto que "posee" el código JavaScript.

El valor de this , cuando se usa en una función, es el objeto que "posee" la función.

El valor de this , cuando se usa en un objeto, es el objeto mismo.

La presente palabra clave en un constructor de objetos no tiene un valor. Es solo un sustituto del nuevo objeto.

El valor de this se convertirá en el nuevo objeto cuando el constructor se utilice para crear un objeto.

Tenga en cuenta que this no es una variable. Es una palabra clave No puedes cambiar el valor de this .

Constructores de JavaScript incorporados

JavaScript tiene constructores incorporados para objetos nativos:

Ejemplo

```
var x1 = new Object();    // A new Object object
var x2 = new String();    // A new String object
var x3 = new Number();    // A new Number object
var x4 = new Boolean();   // A new Boolean object
var x5 = new Array();     // A new Array object
var x6 = new RegExp();    // A new RegExp object
var x7 = new Function();  // A new Function object
var x8 = new Date();      // A new Date object
```

Inténtalo tú mismo "

El objeto Math () no está en la lista. La matemática es un objeto global. La nueva palabra clave no se puede usar en matemáticas.

¿Sabías?

Como puede ver, JavaScript tiene versiones de objeto de los tipos de datos primitivos String, Number y Boolean.

No hay razón para crear objetos complejos. Los valores primitivos se ejecutan mucho más rápido.

Y no hay razón para usar una nueva matriz (). Use los literales de la matriz en su lugar: []

Y no hay ninguna razón para usar RegExp () nuevo. Use literalmente los patrones: / () /

Y no hay razón para usar una nueva función (). Utilice las expresiones de función en su lugar: function () {}.

Y no hay razón para usar un nuevo objeto (). Use los literales de objeto en su lugar: {}

Ejemplo

```
var   x1 = {};          // new object
var x2 = "";           // new primitive string
var x3 = 0;            // new primitive number
var x4 = false;        // new primitive boolean
var x5 = [];           // new array object
var x6 = /()/          // new regexp object
var x7 = function(){};  // new function object
```

Inténtalo tú mismo "

Objetos de cadena

Normalmente, las cadenas se crean como primitivas: var firstName = "John"

Pero las cadenas también se pueden crear como objetos con la nueva palabra clave: var firstName = new String ("John")

Conozca por qué las cadenas no deberían crearse como objeto en el capítulo JS Strings .

Objetos de número

Normalmente, los números se crean como primitivos: var x = 123

Pero también se pueden crear números como objetos usando la nueva palabra clave: var x = número nuevo (123)

Aprenda por qué los números no deben crearse como objeto en el capítulo JS Numbers .

Objetos booleanos

Normalmente, los booleanos se crean como primitivos: var x = false

Pero los booleanos también se pueden crear como objetos usando la nueva palabra clave: var x = new Boolean (false)

Aprenda por qué los booleanos no deben crearse como objeto en el capítulo JS Booleanos .

Los objetos de JavaScript son mutables

Los objetos son mutables: se abordan por referencia, no por valor.

Si la persona es un objeto, la siguiente declaración no creará una copia de la persona:

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

El objeto x no es una copia de la persona. Que es persona. Tanto x como persona son el mismo objeto.

Cualquier cambio en x también cambiará la persona, porque x y persona son el mismo objeto.

Ejemplo

```
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
```

Inténtalo tú mismo "

Nota: las variables de JavaScript no son mutables. Solo objetos JavaScript.

JavaScript for...in Loop

La sentencia JavaScript for ... in bucle a través de las propiedades de un objeto.

Sintaxis

```
for (variable in object) {
    code to be executed
}
```

El bloque de código dentro del bucle for ... in se ejecutará una vez para cada propiedad.

Bucle a través de las propiedades de un objeto:

Ejemplo

```
var person = {fname:"John", lname:"Doe", age:25};

for (x in person) {
    txt += person[x];
}
```

Inténtalo tú mismo "

Agregar nuevas propiedades

Puede agregar nuevas propiedades a un objeto existente simplemente dándole un valor.

Suponga que el objeto de persona ya existe, puede darle nuevas propiedades:

Ejemplo

```
person.nationality = "English";
```

Inténtalo tú mismo "

No puede usar palabras reservadas para nombres de propiedades (o métodos). Se aplican las reglas de denominación de JavaScript.

Eliminando propiedades

La palabra clave delete elimina una propiedad de un objeto:

Ejemplo

```
var person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
delete person.age;   // or delete person["age"];
```

Inténtalo tú mismo "

La palabra clave delete elimina tanto el valor de la propiedad como la propiedad misma.

Después de la eliminación, la propiedad no se puede usar antes de volver a agregarla.

El operador de eliminación está diseñado para usarse en propiedades de objeto. No tiene efecto sobre variables o funciones.

El operador de eliminación no debe utilizarse en propiedades de objeto JavaScript predefinidas. Puede bloquear su aplicación.

JavaScript Object Methods

Los métodos de JavaScript son las acciones que se pueden realizar en los objetos.

Un método JavaScript es una propiedad que contiene una definición de función .

Propiedad	Valor
nombre de pila	John
apellido	Gama
años	50
color de los ojos	azul
nombre completo	function () {return this.firstName + "" + this.lastName;}

Los métodos son funciones almacenadas como propiedades de objeto.

Acceso a métodos de objetos

Crea un método de objeto con la siguiente sintaxis:

```
methodName : function() { code lines }
```

Accede a un método de objeto con la siguiente sintaxis:

```
objectName.methodName()
```

Normalmente describirá fullName () como un método del objeto persona y fullName como una propiedad.

La propiedad fullName se ejecutará (como función) cuando se invoque con ().

Este ejemplo accede al método fullName () de un objeto persona:

Ejemplo

```
name = person.fullName();
```

Inténtalo tú mismo "

Si accede a la propiedad fullName , sin (), devolverá la definición de función :

Ejemplo

```
name = person.fullName;
```

Inténtalo tú mismo "

Agregar nuevos métodos

Agregar métodos a un objeto se realiza dentro de la función del constructor:

Ejemplo

```
function person(firstName, lastName, age, eyeColor) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.eyeColor = eyeColor;
    this.changeName = function (name) {
        this.lastName = name;

  };
}
```

La función changeName () asigna el valor del nombre a la propiedad lastName de la persona.

Ahora puedes probar:

```
myMother.changeName( "Doe");
```

Inténtalo tú mismo "

JavaScript sabe de qué persona estás hablando "sustituyendo" **this** por **myMother** .

JavaScript Object Prototypes

Cada objeto JavaScript tiene un prototipo. El prototipo también es un objeto.

Todos los objetos JavaScript heredan sus propiedades y métodos de su prototipo.

Prototipos JavaScript

Todos los objetos JavaScript heredan las propiedades y los métodos de su prototipo.

Los objetos creados con un objeto literal o con un nuevo objeto () heredan de un prototipo denominado Object.prototype.

Los objetos creados con la nueva fecha () heredan el Date.prototype.

Object.prototype está en la parte superior de la cadena de prototipos.

Todos los objetos JavaScript (Fecha, Matriz, RegExp, Función, ...) heredan de Object.prototype.

Crear un prototipo

La forma estándar de crear un prototipo de objeto es utilizar una función de constructor de objeto:

Ejemplo

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

Con una función de constructor, puede usar la nueva palabra clave para crear nuevos objetos del mismo prototipo:

Ejemplo

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

Inténtalo tú mismo "

La función de constructor es el prototipo para objetos Person.
Se considera una buena práctica nombrar la función de constructor con una primera letra en mayúscula.

Agregar propiedades y métodos a objetos

Algunas veces desea agregar nuevas propiedades (o métodos) a un objeto existente.

A veces desea agregar nuevas propiedades (o métodos) a todos los objetos existentes de un tipo determinado.

A veces, desea agregar nuevas propiedades (o métodos) a un prototipo de objeto.

Agregar una propiedad a un objeto

Agregar una nueva propiedad a un objeto existente es fácil:

Ejemplo

```
myFather.nationality = "English";
```

Inténtalo tú mismo "

La propiedad se agregará a myFather. No a mi madre. No a los objetos de ninguna otra persona.

Agregar un método a un objeto

Agregar un nuevo método a un objeto existente también es fácil:

Ejemplo

```
myFather.name = function () {
    return this.firstName + " " + this.lastName;
};
```

Inténtalo tú mismo "

El método se agregará a myFather. No a mi madre.

Agregar propiedades a un prototipo

No puede agregar una nueva propiedad a un prototipo de la misma manera que agrega una nueva propiedad a un objeto existente, ya que el prototipo no es un objeto existente.

Ejemplo

```
Person.nationality = "English";
```

Inténtalo tú mismo "

Para agregar una nueva propiedad a un prototipo, debe agregarlo a la función de constructor:

Ejemplo

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

Inténtalo tú mismo "

Las propiedades del prototipo pueden tener valores de prototipo (valores predeterminados).

Agregar métodos a un prototipo

La función de su constructor también puede definir métodos:

Ejemplo

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.name = function() {return this.firstName + " " + this.lastName;};
}
```

Inténtalo tú mismo "

Uso de la propiedad prototipo

La propiedad del prototipo JavaScript le permite agregar nuevas propiedades a un prototipo existente:

Ejemplo

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

Inténtalo tú mismo "

La propiedad del prototipo de JavaScript también le permite agregar nuevos métodos a un prototipo existente:

Ejemplo

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}
Person.prototype.name = function() {
  return this.firstName + " " + this.lastName;
};
```

Inténtalo tú mismo "

Solo modifique sus propios prototipos. Nunca modifique los prototipos de objetos JavaScript estándar.