UNIDAD 3

Objetos

IES MARQUES DE COMARES

ÍNDICE:

IES MARQUES DE COMARES

1.- Introducción

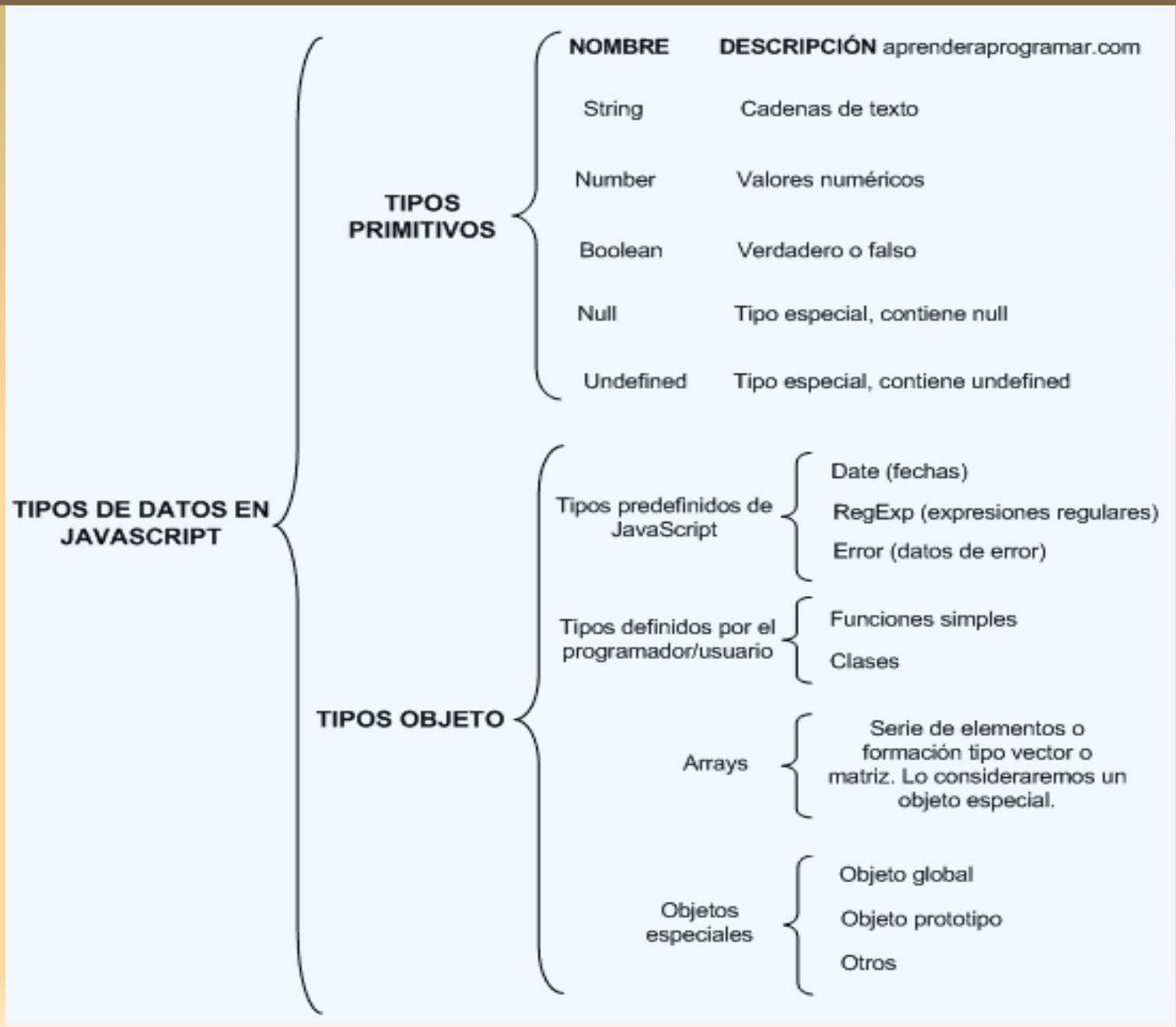
Un objeto podemos definirlo como un ente abstracto usado en programación que permite separar los diferentes componentes de un programa, simplificando así su creación y mantenimiento.

Los objetos integran tanto los procedimientos (métodos) como las variables (atributos) y datos referentes al objeto.

Los objetos se componen de 3 partes fundamentales: metodos, eventos y atributos.

IES MARQUES DE COMARES

1.- Introducción



2.- Objetos: atributos, métodos y eventos

Atributos:
Características o propiedades del objeto que representan características como por ejemplo el color, tamaño, posición, si está o no habilitado, etc.

Métodos:
Son aquellas funciones que permiten modificar los atributos del objeto durante el transcurso del programa. Determinan como va a responder el objeto cuando recibe un mensaje.

Eventos:
Son aquellas acciones mediante las cuales el objeto reconoce que se está interactuando con él, de forma que el objeto se activa y responde al evento.

IES MARQUES DE COMARES

3.- Prototipos

En los lenguajes orientados a objetos, todo objeto pertenece a una clase, por lo que para definir un objeto, es obligatorio primero crear una clase. Además, una clase puede heredar de otra clase, por lo que dispondrá de todas las propiedades y métodos de esa clase de la que hereda.

En Javascript, todos los objetos procedentes del mismo tipo de función constructora, tienen un mismo prototipo (conjunto de métodos y propiedades comunes) con el que enlazan, por lo que, el prototipo es la parte común de todos los objetos del mismo tipo.

Para obtener el prototipo de un objeto podemos usar el método getPrototypeof que es un método de la clase genérica Object.
Para acceder al prototipo de una propiedad podemos usar la propiedad –proto--

IES MARQUES DE COMARES

Prototipos

Un prototipo es un objeto del que otros objetos heredan propiedades, y los objetos siempre heredan propiedades de algún objeto anterior, de este modo solo el objeto original y primigenio (Object) de javascript es el único que no hereda de nadie.

Por tanto, los objetos creados por ejemplo de forma literal o new Objetc(), heredan directamente del objeto Object.prototype.

Mientras que los objetos creados, por ejemplo, con new Date(), heredan de Date.prototype.

Es el objeto **Object.prototype** el primer eslabón de la cadena, de este modo todos los objetos de JS heredan de él, ya sean arrays, fechas, funciones…

IES MARQUES DE COMARES

Prototipos

Javascript es un lenguaje orientado a objetos basado en prototipos y no en clases. Cuando en javascript implementamos una clase, en realidad, el motor del navegador transforma esa clase en un prototipo.

Un prototipo no es más que un mecanismo por el cual un objeto puede heredar propiedades y métodos de un objeto padre. De hecho, la herencia en javascript se da en la cadena de prototipos.

Cada objeto tiene su prototipo. Veamos algún ejemplo que aclare esto:

```
const vehiculo1 ={
    nombre: "coche",
    //notación tradicional ES5
    acelerar:function() { console.log("esto es aumentar velocidad");}
    }
const vehiculo2 ={
    nombre: "motocicleta",
    //notación ES6
    acelerar() { console.log("esto es aumentar velocidad");}
    }
console.log(vehiculo1);
console.log(vehiculo2);
```

IES MARQUES DE COMARES

Prototipos

Ahora bien, la creación de objetos como hemos hecho en la diapositiva anterior no es viable, porque cada vez que vayamos a crear un tipo de vehículo habría que copiar prácticamente el código del objeto con su propiedad y método. Lo ideal sería poder crear un prototipo a partir del cual generemos multitud de vehículos.

Por tanto, en la siguiente diapositiva veremos que lo que hay que hacer es crearse una función denominada constructora (porque solo hay que construirla una vez) y a partir de ella generaremos nuevos objetos del tipo de función constructora

IES MARQUES DE COMARES

Prototipos

//para pasar propiedades a la función constructora hay que hacerlo como parámetros.

```
function Vehiculo (nombre,tipo){
// atributos
this.nombre=nombre;
this.tipo=tipo;
// métodos
this.acelerar=function(){console.log("Esto es aumentar la velocidad")};
}

const   coche= new Vehiculo("Coche","Motor"),
        motocicleta=new Vehiculo("Motocicleta","Motor");
console.log(coche);
console.log(motocicleta);
```

IES MARQUES DE COMARES

Prototipos

El problema que tendría el prototipo anterior es que al crear dos objetos de dicho prototipo, se está repitiendo el método acelerar en ambos vehículos, y, como cada objeto está almacenado en una variable, ocuparía memoria. Por tanto, lo mejor sería sacar al método acelerar del prototipo, para reutilizarlo de una manera más eficiente y mejorar el rendimiento de la aplicación. Entonces lo ideal sería asignarle el método acelerar y asignárselo al prototipo de la función Vehículo

IES MARQUES DE COMARES

3.1 Modificar Prototipos

Para modificar un prototipo hay que indicar la propiedad prototype y después definir las propiedades y métodos deseados. Solo está disponible en las funciones constructoras.

Veamos como asignar los métodos al prototipo de la función y no al del vehículo

```
function Vehiculo (nombre,tipo){
// atributos
this.nombre=nombre;
this.tipo=tipo;
}
// métodos agregados al prototipo de la función constructora
y no al prototipo de vehículo
Vehiculo.prototype.acelerar=function(){console.log("Esto es aumentar la veloci
dad")};
const   coche= new Vehiculo("Coche","Motor"),
        motocicleta=new Vehiculo("Motocicleta","Motor");
console.log(coche);
console.log(motocicleta);
coche.acelerar();
motocicleta.acelerar();
```

IES MARQUES DE COMARES

Veamos otro ejemplo:

```
//Función constructora
 function Punto(coordX,coordY){
       this.x=coordX;
       this.y=coordY;
       this.mostrarCoordenadas=function(){
          console.log(this.x);
          console.log(this.y);
          }
    }
    let a=new Punto(10,20);
    let b=new Punto(-3,6);

    console.log(a.mostrarCoordenadas());
    console.log(b.mostrarCoordenadas());
```

Ya sabemos que no eficiente porque tenemos el método mostrarCoordenadas dentro del prototipo Punto y deberíamos sacarlo al prototipo padre.

IES MARQUES DE COMARES

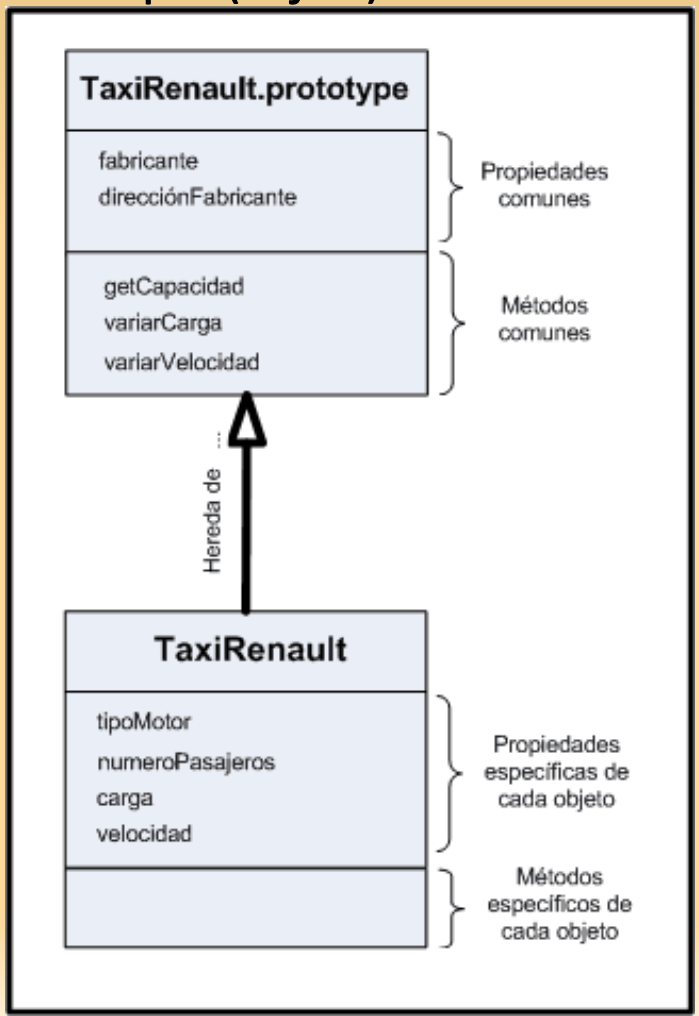
```
// definamos un método y propiedad para el prototipo objetopadre del objeto Punto
Punto.prototype.sumaXY=function(){
        Return this.x+this.y;
        }
console.log(Punto.prototype);
console.log(a.sumaXY);
```

Modificar Prototipos

Incluso podremos modificar el prototipo de un objeto estándar.
Por ejemplo, modifiquemos el objeto Array para añadirle el método obtenerPares, de forma que de ahora en adelante ese método podemos utilizarlo con cualquier Array.

3.2- Herecia con Prototipos

La idea de la herencia basada en prototipos JavaScript es definir un objeto (prototipo) donde se aloja toda la información común que comparten todos los objetos de ese tipo (hijos).

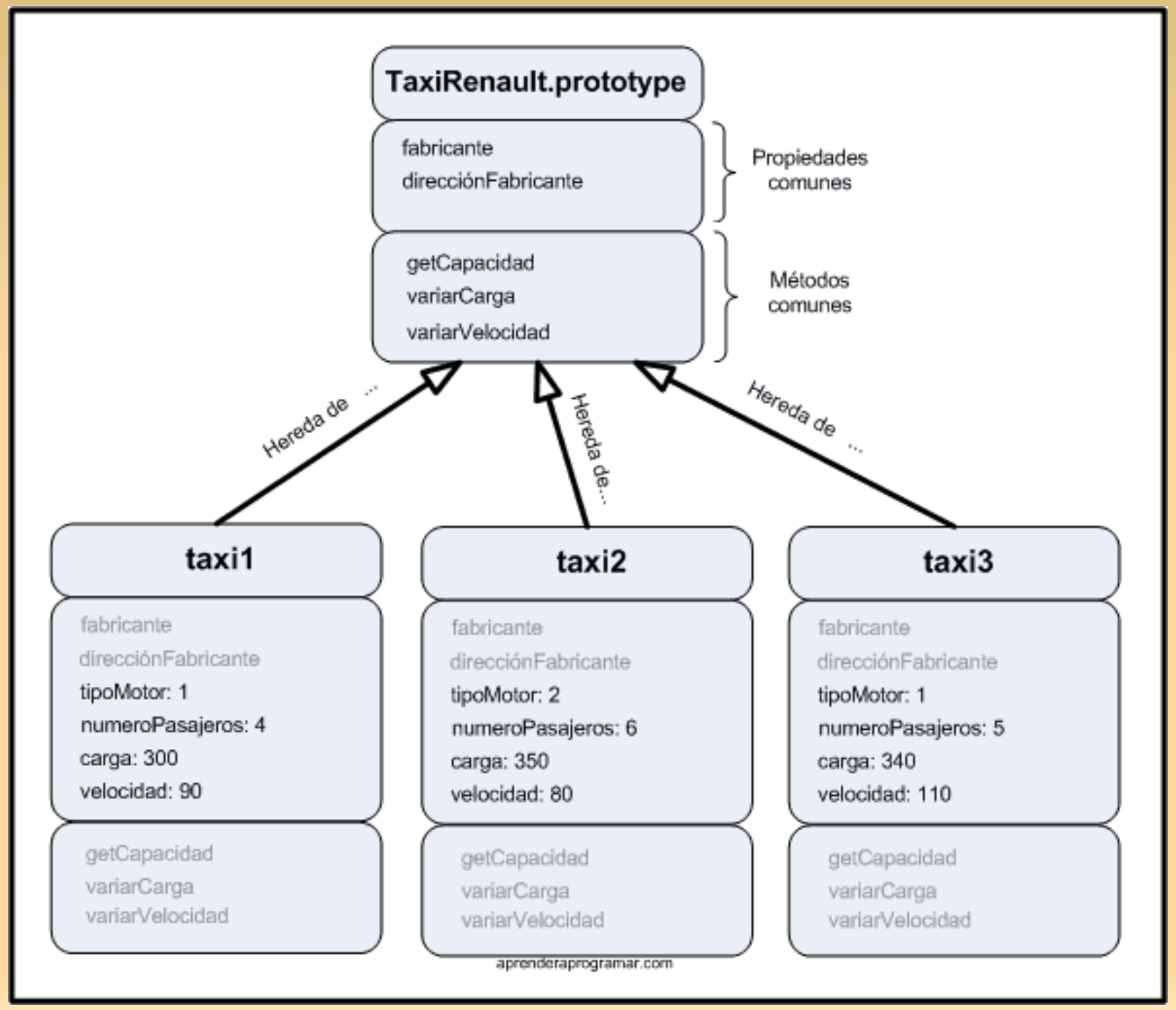


3.2- Herencia con Prototipos

Cuando un objeto invoca una propiedad, por ejemplo taxi1.fabricante, se comprueba si dicha propiedad está definida como propiedad específica del objeto taxi1, y si no es así, se busca en su prototipo de modo que si la propiedad existe en el prototipo, se devuelve esa propiedad de la misma manera que si fuera una propiedad del objeto. Si no se encontrara en el prototipo, se buscaría en el prototipo del prototipo (el padre del padre…) y así sucesivamente hasta encontrarla o no, denominándose a este proceso “cadena de prototipos (prototype chain)”

IES MARQUES DE COMARES

3.2- Herecia con Prototipos



IES MARQUES DE COMARES

Herecia con Prototipos

Ejemplo de herencia con prototipos

```
function Animal(nombre,especie){
    // atributos
    this.nombre=nombre;
    this.especie=especie;
    }
// métodos agregados al prototipo de la función y no al prototipo de vehículo
Animal.prototype.identificar=function(){console.log("Emito mi sonido característico")};
Animal.prototype.crecer=function(){console.log("con los años me voy haciendo mayor");}
 //herencia en prototipos
//generamos una función constructora para crear el coche audi por ejemplo
function Gato(nombre,especie,subespecie){
//crear variable que vamos a llamar super que haría lo que super en clases para asignarle cual es el prototipo padre
this.super=Animal;
//Ejecutamos super con los parámetros del prototipo Animal
this.super(nombre,especie);
//añadimos el atributo subespecie del prototipo Gato
this.subespecie=subespecie;
}
//Gato hereda de Animal
Gato.prototype=new Animal();
//ahora hacemos que el constructor obtenga todas las propiedades del prototipo padre Animal, así coche hereda todo lo de Animal
Gato.prototype.constructor=Gato;
//podemos sobreescribir métodos del padre"
Gato.prototype.identificar=function(){console.log("miau, miau");}
//nuevo método
Gato.prototype.reproducir=function(){console.log("Soy vivíparo");}
//ya podemos decir que gato es un objeto del prototipo Gato;
const   gato= new Gato("Garfield","Mamífero","Felino"),
        pato=new Animal("Lucas","Ave");
console.log(gato);
console.log(pato);
```

IES MARQUES DE COMARES

Herecia con Prototipos

Veamos como se heredan los métodos, sabiendo que el método identificar lo hemos sobreescrito en Gato, pero el método crecer no lo tenemos en Gato, pero lo tenemos en el prototipo del objeto padre que es Animal

```
console.log(gato);
//se ejectua el método del prototipo Gato
console.log(gato.reproducir);
//se ejecuta el método del prototipo Gato sobreescrito de
Animal
console.log(gato.identificar);
//se ejecuta el método del prototipo padre de Gato (porque
 no está en Gato), es decir, de Animal
console.log(gato.crecer);
```

IES MARQUES DE COMARES

3.3 Definir prototipos

Ejemplo:

```
<!DOCTYPE html>
<head>
   <meta charset="UTF-8">
   <title>Definir prototipost</title>
   <script>
   function TaxiRenault (tipoMotor, numeroPasajeros, carga, velocidad) {
      this.tipoMotor = tipoMotor;
      this.numeroPasajeros = numeroPasajeros;
      this.carga = carga;
      this.velocidad = velocidad;
      }
TaxiRenault.prototype.fabricante = 'Renault, S.A.';
TaxiRenault.prototype.direccionFabricante = 'c/R, Paris';
TaxiRenault.prototype.getCapacidad = function () {
   if (tipoMotor == 'Diesel')
      return 40;
   else
      return 35;
   }
TaxiRenault.prototype.variarCarga = function (variacion) {
    this.carga = this.carga + variacion;
   }
TaxiRenault.prototype.variarVelocidad = function (variacion) {
   this.velocidad = this.velocidad + variacion;
   }
function ejemploObjetos() {
var taxi1 = new TaxiRenault(1, 4, 300, 90);                var taxi2 = new TaxiRenault(2, 6, 350, 80);                var taxi3 = new TaxiRenault(1, 5, 340, 110);
console.log('El fabricante del taxi 2 es ' + taxi2.fabricante + ' y la velocidad del taxi 2 es ' + taxi2.velocidad);
taxi2.variarVelocidad(-10);
console.log('El taxi 2 ha reducido su velocidad y ahora es ' + taxi2.velocidad);
}
ejemploObjetos();
</script>
</head>
</body>
</html>
```

4.- Creación de objetos

Todos los objetos en JavaScript provienen de Object; todos los objetos heredan métodos y propiedades de Object.prototype, aunque pueden ser sobrecargados. Sin embargo, un Objeto puede ser deliberadamente creado para que esto no sea cierto (por ejemplo usando Object.create(null)), o bien alterado para que no cumpla esta propiedad (por ejemplo usando Object.setPrototypeOf).

La forma básica

La forma más sencilla es crear el objeto y sus propiedades en un solo paso (objeto literal), del modo siguiente.

```
var = libro{
páginas: 18,
tema: “Infantil”,
tapa: “dura”
};
```

El inconveniente es que hay que tener **claras las propiedades desde el inicio** porqué no podremos añadir nuevas propiedades, pero es la más recomendable para empezar a programar en orientación a objetos.

IES MARQUES DE COMARES

Ejemplo más completo:

```
var hotel = {
nombre: 'Hotel mediaestrella’,
habitaciones: 137,
ocupadas: 30,
tipo habitaciones: ['sencilla', 'doble', 'suite’],
wifi: true,
verificaDisponibilidad: function(){
        return this.habitaciones-this.ocupadas;
        }
};
```

La forma compleja

Se suele usar cuando **no sabemos cuántas propiedades vamos a tener**. Podemos usar la notación punto o la de corchetes.

```
var libro = new Object();
libro.paginas = 200;
libro.tema= “novela”;
libro.tapa= “blanda”;
```

```
var libro = new Object();
libro[“paginas”] = 200;
libro[“tema”]=“novela”;
libro[“tapa”]=“blanda;
```

La ventaja es que podemos crear propiedades nuevas en cualquier momento, pero, tiene el inconveniente de que puede confundirnos más por el desorden que puede llegar a alcanzar un objeto.

Nota: si hay espacio en el nombre del atributo, hay que utilizar obligatoriamente la notación corchetes.

Modificación de objetos

Es posible modificar un objeto añadiendo y/o modificando propiedades tanto para los que hayamos creado nosotros como para los Objeto internos.

Ejemplo:

nuevo método para la clase Date() llamado 'getCadenaFecha()', que nos devuelve la fecha en una cadena de texto en formato dd/mm/yyyy:

```
Date.prototype.getCadenaFecha = function(){
        var dia  = String( this.getDate() );
        var mes  = String( this.getMonth() );
        var anno = this.getFullYear();
if( dia.length == 1 ) {  dia = "0" + dia;   }
if( mes.length == 1 ) {mes = "0" + mes;  }
return dia + "/" + mes + "/" + anno;
        }
```

Para invocarla:

```
var objFecha = new Date( 2011, 4, 3 );
alert("Fecha:  [" + objFecha.getCadenaFecha() + "]" );
```

5.- Acceder a propiedades y métodos

Para acceder a una propiedad o método de un objeto, se usa el nombre del objeto seguido de un punto (.) y a continuación el nombre de la propiedad o método al que se desea acceder. Ej.:

```
var hotelNombre = hotel.nombre;
var habitacionesLibres = hotel.verificaDisponibilidad();
```

Para acceder a una propiedad de un objeto se usan corchetes. Ej.:

```
var hotelNombre = hotel['nombre'];
```

Acceder a propiedades y métodos

Podemos recorrer las propiedades y/o métodos de un objeto mediante un bucle for.

Ejemplo:

```
var unObjeto = {numemp: 1, apellnom: “Pepe Pérez”};

 for( propiedad in unObjeto ) {
        document.write( propiedad + " = " + unObjeto[propiedad] + "<br />");
        }
```

6.- Objetos predefinidos en javascript

Javascript posee de forma predefinida una serie de tipos por referencia (objetos) cuya funcionalidad podemos usar.

Los tipos por referencia que posee Javascript son:

- Array
- String
- Date
- Math
- Boolean
- Number
- RegExp

Ver explicación de cada objeto en el siguiente enlace:
http://www.cs.us.es/cursos/mp/temas/Web-tema-09.pdf

IES MARQUES DE COMARES

7.- Clases en javascript (azúcar sintáctico)

Las clases en javascript fueron introducidas en ECMAScript 2015, como mejora sintáctica sobre la herencia basada en prototipos. Sin embargo, **no** introduce un nuevo modelo de herencia orientada a objetos, sino que provee una sintaxis mucho más clara y simple para crear objetos y la herencia.

Para declarar una clase, se utiliza la palabra reservada class. A diferencia de Java, solo pueden tener un constructor definido con la palabra “constructor”.

El contenido (cuerpo) de una clase es la parte que se encuentra entre las llaves {}, donde se definen el constructor, propiedades y métodos.

Un constructor puede usar la palabra reservada super para llamar al constructor de una superclase

IES MARQUES DE COMARES

7.- Ejemplo clase Animal hecha antes con prototipos

```
class Animal{
    //constructor definido con la palabra constructor
    constructor (nombre,especie){
        this.nombre=nombre;
        this.especie=especie;
        }
    // métodos de la clase, se definen de forma literal, es decir, de
 la forma nombremetodo(){}
    identificar(){console.log("Emito mi sonido característico");}
    crecer(){console.log("con los años me voy haciendo mayor");}
    }
    const   gato= new Animal("Garfield","Mamífero"),
            pato=new Animal("Lucas","Ave");
    console.log(gato);
    console.log(pato);
    gato.identificar();
```

Por tanto, la herencia simplifica la creación de objetos en javascript.

IES MARQUES DE COMARES

Otro Ejemplo: Clase circunferencia

```
class Circunferencia{
        //constructor
        constructor(radio){
                this.radio=radio;
        }
        //método para calcular el área de la circunferencia
        calculoArea () {
                document.write("El área de la circunferencia de radio "+this.radio+" es "+Math.PI*this.radio*this.radio+"<br>");
          }
        //método para calcular longitud de la circunferencia
        calculoLongitud(){
                document.write("La longitud de la circunferencia de radio "+this.radio+" es "+ 2*Math.PI*this.radio+"<br>");
        }
}
```

Herencia con clases

Mediante “extends” podemos usar herencia con clases en javascript.

Ejemplo: Esfera una circunferencia de cierto material con volumen.

```
class Circunferencia{
		//constructor
		constructor(radio){	this.radio=radio;	}
		//método para calcular el área de la circunferencia
		calculoArea () {
		document.write("El área de la circunferencia de radio "+this.radio+" es "+Math.PI*this.radio*this.radio+"<br>");
		 }
		//método para calcular longitud de la circunferencia
		calculoLongitud(){
		document.write("La longitud de la circunferencia de radio "+this.radio+" es "+ 2*Math.PI*this.radio+"<br>");
		}
}
class Esfera extends Circunferencia {
		constructor(radio){
				super(radio);
		}
		material= prompt("Inroduzca material de la esfera");
		calculoVolumen(material){
		document.write("El volumen de la esfera de "+ this.material + " de radio "+this.radio+ " es
"+4/3*Math.PI*this.radio*this.radio*this.radio);
		}
}
```

IES MARQUES DE COMARES

Ejemplo anterior con Herencia

```
//herencia
class Gato extends Animal{
    constructor(nombre,especie,subespecie){
    //invocacion método super para llamar al constructor de la clase de
 la que hereda
    super(nombre,especie);
    this.subespecie=subespecie;
    }
    // sobreescritura de método identificar
    identificar(){console.log("miau miau");}
    // nuevo método
    reproducir(){console.log("Soy vivíparo");}
}
const gato= new Gato("Garfield","Mamífero","Felino");
console.log(gato);
gato.identificar();
gato.crecer();
gato.reproducir();
```

IES MARQUES DE COMARES

Notas importante en javascript

Al ser javascript un lenguaje basado en prototipos, no existen tipos de clases, sino que todas las clases son públicas.

Aunque se pueden usar métodos estáticos por ejemplo, no tienen sentido al ser un lenguaje basado en prototipos. Por cierto, los métodos get y set son tratados como atributos (propiedades) y no como métodos aunque realmente son funciones (métodos)

IES MARQUES DE COMARES

Implementación y uso get y set

```
class Animal{
    //constructor definido con la palabra constructor
    constructor (nombre,especie){
        this.nombre=nombre;
        this.especie=especie;
        }
    // métodos de la clase, se definen de forma literal, es decir, de la forma nombremetodo(){}
    identificar(){console.log("Emito mi sonido característico");}
    crecer(){console.log("con los años me voy haciendo mayor");}
    }
//herencia
class Gato extends Animal{
    constructor(nombre,especie,subespecie){
    //invocacion método super para llamar al constructor de la clase de la que hereda
    super(nombre,especie);
    this.subespecie=subespecie;
    this.raza=null;
    }
    // sobreescritura de método identificar
    identificar(){console.log("miau miau");}
    // nuevo método
    reproducir(){console.log("Soy vivíparo");}
    //métodos get y set para obtener y modificar la propiedad raza
    get getRaza(){return this.raza};
    set setRaza(raza){this.raza=raza};
}
const gato= new Gato("Garfield","Mamífero","Felino");
gato.setRaza="Pardo";
console.log(gato.getRaza);
```

IES MARQUES DE COMARES

8.- Objetos Literales

Las nuevas características de Javascript permiten una nueva forma de escribir atributos y métodos, así como asignarlos.

Ej:

Método clásico

```
let nombre="Pepe";
let email="pepe@inventado.com";
//Javascirpt sabe y asigna de forma automática a los atributos el valor de las variables
const alumno{
 nombre: nombre,
 email: email,
 matricular: function (){
    console.log("alumno matriculado");
    }
}

console.log(alumno);
alumno.matricular();
```

Método nuevo

```
let nombre="Pepe";
let email="pepe@inventado.com"
const alumno{
nombre,
email,
matricular(){
   console.log("alumno matriculado");
   }
}

console.log(alumno);
alumno.matricular();
```

IES MARQUES DE COMARES

9.- Destructuración con objetos

Nos va a permitir asignar propiedades de un objeto a variables.

Ej:

```
const alumno={
        nombre: “Pepe”,
        apellidos: “Gil Robles”,
        email: pepe@inventado.com
}

let {nombre,apellidos,email}=persona;
console.log(nombre);
console.log(apellidos);
console.log(email);
```