# U2. Objetos

Desarrollo Web en Entorno Cliente

# Índice

- Objetos
  - ¿Qué son?
  - Crear objetos
    - Con Object
    - Con literales
    - Con constructor
  - Modificar objetos
  - Herencia
  - Recorrer objetos
  - Objetos nativos de Javascript

# ¿Qué son?

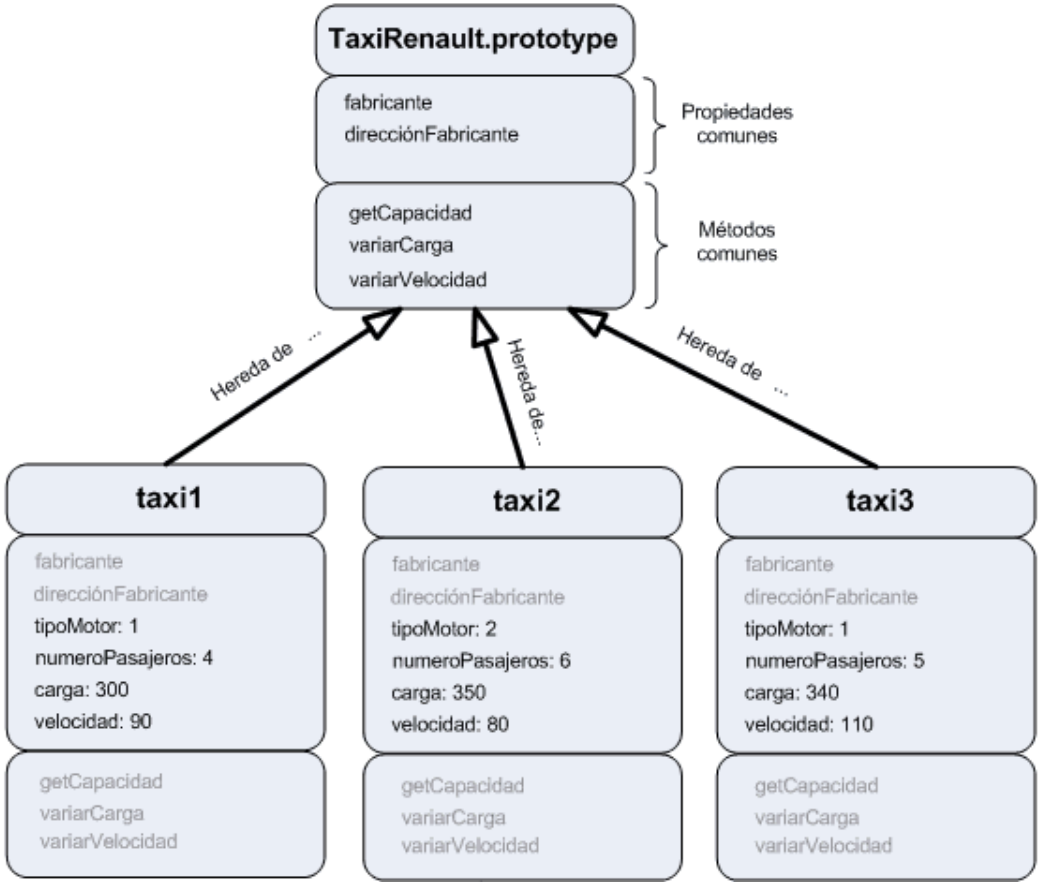
- La programación orientada a objetos es un paradigma de programación que utiliza la abstracción para crear modelos que permitan agilizar la generación de código.
- Terminología
  - Clase: Define las características del Objeto.
  - Objeto: Una instancia de una Clase.
  - Propiedad: Una característica del Objeto, como el color.
  - Método: Una capacidad del Objeto, como caminar.
  - Constructor: Es un método llamado en el momento de la creación de instancias.
  - Herencia: Una Clase puede heredar características de otra Clase.

# ¿Qué son?

- JavaScript proporciona un estilo de programación orientada a objetos basada en prototipos no en clases.
- En JS se crean objetos nuevos por clonación de otros que hacen de prototipos.
- La definición de un objeto que hace de prototipo en JS es aquel que posee una función que hace de constructor.

# ¿Qué son?

- Un objeto son un conjunto de:
  - Propiedades (atributos)
  - Métodos (funciones)
- Los objetos que se basan en el prototipo (equivale a la clase), heredan parte de sus características y pueden variar en otras.



# Crear objetos: nuevos

- **Opción 1**: podemos crear objetos con `Object()` y crear propiedades y métodos.
  - Ejemplo: Creamos gato con las propiedades nombre y los métodos maulla y ronrronea

```
var gato = new Object();
gato.nombre = “Félix”;
gato.maulla = function(){
        alert(“Miau miau…”);
}
gato.ronrronea = function(){
        alert(“rrrrrr…”);
}
```

  - Usamos gato: gato.nombre; gato.maulla();

# Crear objetos: nuevos

- **Opción 2**: JS admite el uso de literales para crear objetos.
- Ejemplo

```
var miGato = {
    nombre: "Félix", //separo con , los métodos y atributos
    ronrronea: function(){
     alert("rrrrr…");
    }
}
```

# Crear objetos: nuevos

- **Opción 3**: podemos crear objetos nuevos creando constructores. La función **constructor** cumple:
  - Por convención empieza por mayúsculas.
  - Emplea this
  - No devuelve nada

- Ejemplo constructor

```
function Gato(nombre){
        this.nombre = nombre;
        this.ronrronea = function(){
            alert("rrrrrr…");
        }
}
```

- Creamos un objeto con el constructor: var Felix = new Gato("Félix");

# Aclaración this

- Para acceder a las propiedades dentro de un objeto prototipo empleamos la palabra reservada **this**.
  - This siempre toma el valor de un objeto
  - This apunta al objeto contexto de la función donde se ejecuta:
    - Si la función se ejecuta como global, "this" será el propio objeto global.
    - Si la función se ejecuta como método de un objeto, entonces "this" es el objeto que está recibiendo este método.

# Ejemplos crear objetos

- Ejemplo:

```
<script type="text/javascript">
  function Coche(marca_in, modelo_in, anyo_in){
    this.marca = marca_in;
    this.modelo = modelo_in;
    this.anyo = anyo_in;
  }
</script>
```

# Ejemplos crear objetos

- Es posible añadir otras propiedades a cada instancia del objeto, por ejemplo:

```
function Coche (marca_in, modelo_in, anyo_in){
  this.marca = marca_in;
  this.modelo = modelo_in;
  this.anyo = anyo_in;
}
var mi_coche = new coche("Pegeout", "206cc", "2003");
mi_coche.color = "azul";
```

# Ejemplos crear objetos

- Una vez declarado el nuevo tipo de objeto se pueden crear instancias mediante la palabra clave `new`:

```
<script type="text/javascript">
  var coches = new Array(4);
  coches[0] = new Coche("Ferrari", "Scaglietti", "2010");
  coches[1] = new Coche("BMW", "Z4", "2010");
  coches[2] = new Coche("Seat", "Toledo", "1999");
  coches[3] = new Coche("Fiat", "500", "1995");
  for(i=0; i<coches.length; i++){
    document.write("Marca: "  + coches[i].marca +
    " - Modelo: " + coches[i].modelo + " - A&ntilde;o
    de fabricaci&oacute;n: " + coches[i].anyo + "<br>");
  }
</script>
```

# Modificar objetos

- Una vez creado un constructor, podemos modificarlo con prototype de la siguiente manera:

```
Gato.prototype.especie = “felino”; //creo una propiedad
Gato.prototype.come = function() {
    console.log(“ñam!”);
} //creo un método
```

- El objeto creado con constructor, recuerda quién es su padre (tiene la propiedad **constructor**) y adquiere los nuevos métodos y propiedades generados a posteriori.

  Felix.especie y Felix.come() existen a partir del cambio del constructor Gato.

- Si en cambio añado propiedades o métodos en una instancia, sólo lo tiene esa instancia. Ejemplo:

```
Felix.edad = 2; //Añado la propiedad edad sólo a Félix porque no lo hago con su constructor
```

# Herencia

- Podemos crear un objeto a partir de otro empleando call() o apply()
- Por ejemplo creamos Gato a partir de Animal:

```
function Animal(nombre){
    this.nombre = nombre;
}

function Gato(nombre, especie){
    Animal.call(this,nombre); //Animal.apply(this,[nombre]);
    this.especie = especie;
    this.maulla = function(){ alert("Miau");}
}
```

# Recorrer objetos

- Podemos acceder a las propiedades de un objeto con la sintaxis: **objeto.propiedad** o **objeto[“propiedad”].**
  - Aclaración: El nombre de la propiedad de un objeto puede ser cualquier cadena válida de JavaScript. Cualquier identificador no válido, requerirá la notación de corchetes.
- Podemos consultar sus propiedad con **Object.keys()**
- **Forin** permite recorrer el objeto sin saber el nombre de los campos

# Recorrer objetos

- for in permite recorrer todas las propiedades del objeto:

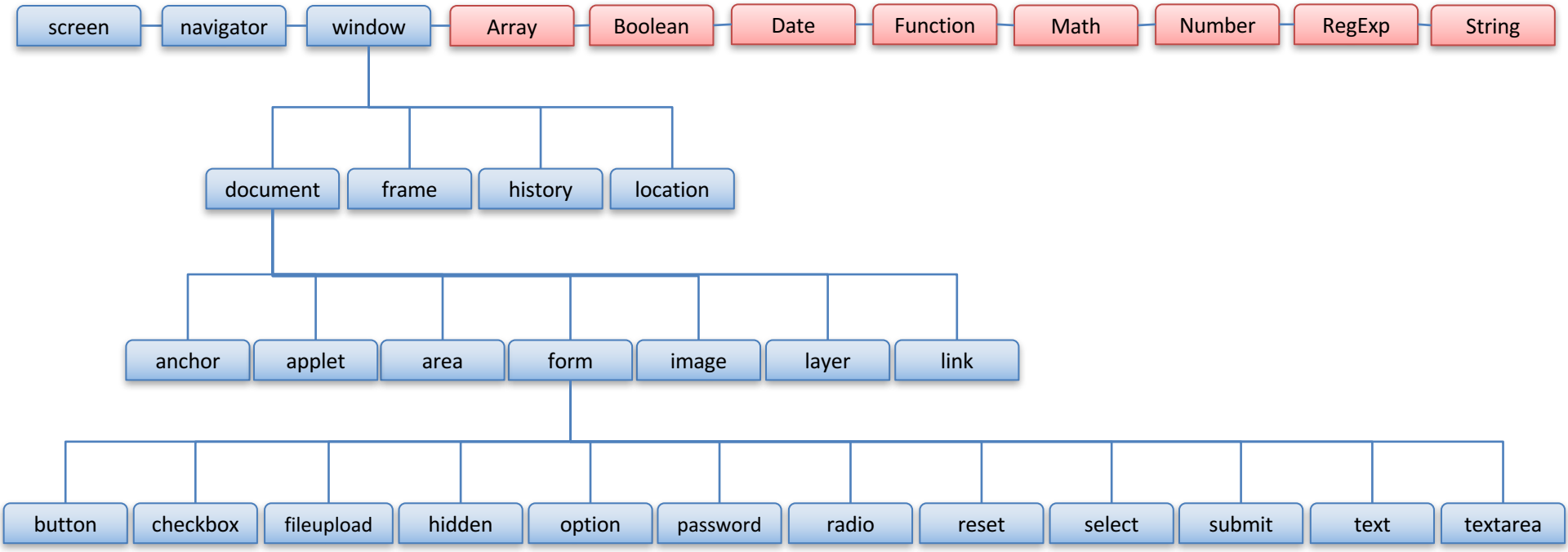
```
for (variable in objeto) {
  sentencias
} //variable tomará los valores de las propiedades y métodos del objeto
```

- Ejemplo:

```
var person = {fname:"John", lname:"Doe", age:25};
for (var x in person) {
    console.log(person[x]);
}
```

# Objetos nativos de JavaScript

- Los objetos de JavaScript se ordenan de modo jerárquico.



# Crear objetos: objetos nativos

- JS proporciona una amplia colección de **objetos que ya existen (nativos)**, a partir de los cuáles podemos crear objetos y usar sus propiedades y métodos. Ejemplo:

```
var fecha = new Date(); //fecha es la fecha del día de hoy
```

- Accedo a las propiedades del objeto creado usando
  - fecha.nombrePropiedad
  - fecha.nombreMetodo([parámetros])
    - Ejemplo: var anyo = fecha.getFullYear(); //anyo es año actual