JavaScript Prototypes es uno de los concepto de JavaScript menos entendidos por parte de los desarrolladores. No es difícil de extrañar, ya que si JavaScript destaca por algo es por sus peculiaridades. Vamos a intentar acercarnos a este concepto a través de un ejemplo,para ello construimos dos objetos Persona.

```
function Persona(nombre,edad) {

this.nombre=nombre;

this.edad=edad;

this.info=function() {

console.log(this.nombre+" " +this.edad);

}

}

var pepito= new Persona("pepito",20);

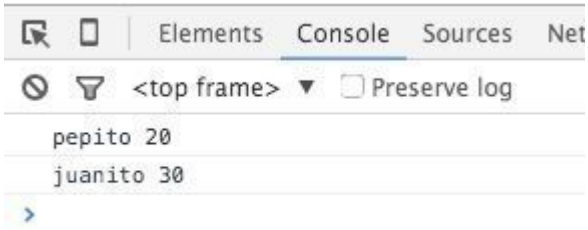
pepito.info();

var juanito=new Persona("juanito",30);

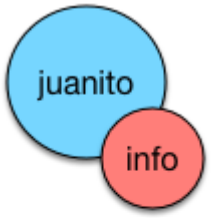
juanito.info();
```

Acabamos de usar JavaScript para crear dos objetos apoyandonos en una función constructora. Nada mas crearlos invocamos al método info() e imprimimos la información

por pantalla.



Todo parece funcionar correctamente , sin embargo hay un pero, resulta que la función info() pertenece a cada uno de los objetos y está duplicada.





Esto es fácil de comprobar ya que podemos cambiar el código por el siguiente:

```
function Persona(nombre,edad) {

this.nombre=nombre;

this.edad=edad;
```

```
this.info=function() {

console.log(this.nombre+" " +this.edad);

}

}

var pepito= new Persona("pepito",20);

pepito.info();

var juanito=new Persona("juanito",30);

juanito.info=function() {

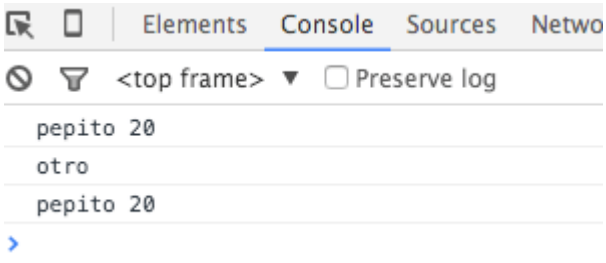
console.log("otro");

}

juanito.info();

pepito.info();
```

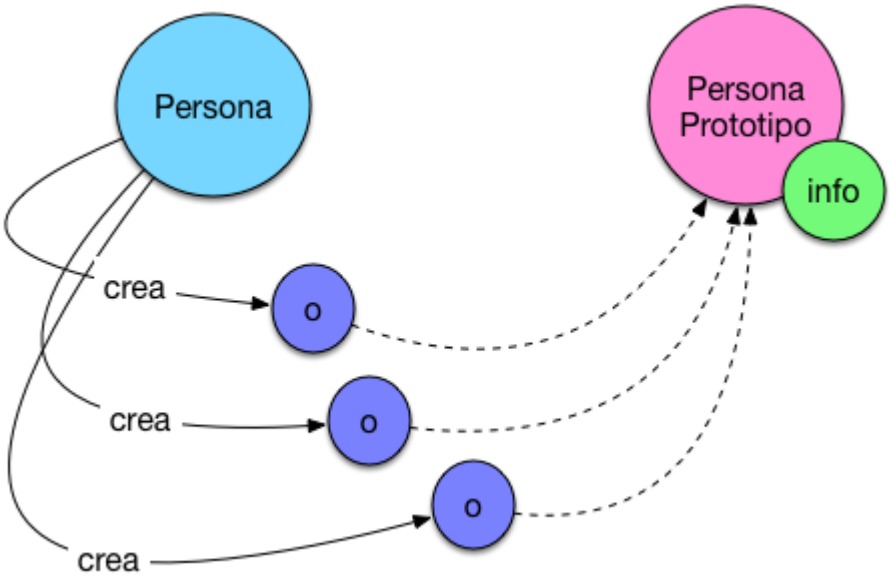
En este caso estamos cambiando la implementación de la función info() pero estos cambios solo afectan a un objeto el otro se comporta de la forma habitual.



Así pues tenemos dos funciones ya que hemos construido dos objetos. Cuando construyamos 1000 cada uno tendrá su propia función y nos sorprenderemos de que JavaScript vaya lento.

JavaScript Prototypes

Para solventar este problema debemos utilizar JavaScript Prototypes. ¿Cómo funcionan y que son?. Cada “clase” de JavaScript que construimos dispone de su propio prototipo. Un prototipo no es ni mas ni menos que un objeto . Este objeto es clonado por los nuevos objetos que creamos con nuestra clase.



Así pues en nuestro caso disponemos de la clase “Persona” que crea los diferentes objetos y lo que queremos es que todos ellos compartan la misma función info. El código correcto para realizar la operación será:

```
function Persona(nombre, edad) {

 this.nombre = nombre;
 this.edad = edad;

 }

 Persona.prototype.info = function() {

 console.log(this.nombre + " " + this.edad);

 }


 var pepito = new Persona("pepito", 20);
 var juanito = new Persona("juanito", 30);

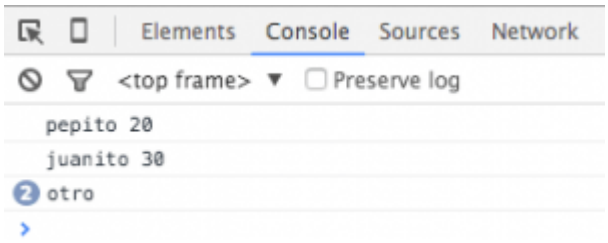
 pepito.info();
 juanito.info();


 Persona.prototype.info = function() {

 console.log("otro");
 }
```

```
juanito.info();
pepito.info();
```

De esta manera almacenamos el método info() en el objeto de prototipo, si cambiamos el comportamiento del método a nivel de prototipo todos los objetos se verán afectados.



La flexibilidad de JavaScript es infinita.

Otros artículos relacionados: JavaScript Module Pattern , JavaScript Reactive Programming