El prototipo de una clase existe para que le podamos dar al mismo método, en lugar de dárselas a una clase creada

De este modo, cuando creas objetos en base a ea clase no sobrecargas los recursos de la maquina con muchísimos métodos en cada objeto de clase, sino que el método que querés ya va a estar guardado en su prototipo

function crearUsuario() {

  // Crea una Clase de ES6 o una función constructor llamada "Usuario"

  // Debe aceptar un objeto "opciones" con las propiedades "usuario", "nombre", "email" y "password"

  // En el `contructor`, define el usuario, el nombre, el email y la contraseña

  // El `contructor` debe tener un método llamado "saludar" en su `prototype` que devuelva una string 'Hola, mi nombre es {{nombre}}'

  // {{nombre}} debe ser el nombre definido en cada instancia

  // Devuelve la clase

  // Tu código:  var opciones = new Usuario();

A la clase no le instanciamos usuario ni nombre, etc (propiedades) por eso el .this. Se lle referenciaran al objeto que llamemos por argumento cuando creemos un nuevo objeto a partir de la clase ej: new Usuario(elloco) . this referencia al objeto que en ese momento invoca al funcion constructor

function Usuario(opciones) {

this.usuario = opciones.usuario;

this.nombre = opciones.nombre;

this.email = opciones.email;

this.password = opciones.password;

}

Usuario.prototype.saludar = function() {

return 'Hola, mi nombre es ' + this.nombre;

}

return Usuario;

}

Esta es la forma de crear función constructora …la otra sería con sería clase ES 6

class Usuario {

constructor (opciones) {

this.usuario = opciones.usuario;

this.nombre = opciones.nombre;

this.email = opciones.email;

this.password = opciones.password;

}

}

Usuario.prototype.saludar = function() {

return 'Hola, mi nombre es ' + this.nombre;

}

return Usuario;

}

Pododria agregar l método dentro de la clase asi

this.saludar = function() {

return 'Hola, mi nombre es ' + this.nombre;

}

function agregarStringInvertida() {

  // Agrega un método al prototype de String que devuelva la misma cadena de caracteres, pero invertida.

  // El método debe llamarse "reverse"

  // Ej: 'menem'.reverse() => menem

  // 'toni'.reverse() => 'inot'

  // Pista: Necesitarás usar "this" dentro de "reverse"

  String.prototype.reverse = function() {

   var stringinvertida = '';

   for (i = this.length -1; i >= 0; i--) {

stringinvertida = stringinvertida + this.charAt(i);

   }

return stringinvertida

  }

}

Citamos el prototipe de string y le agregamos un método… abrimos la funcion y creamos una variable que será el string invertido que devolveremos… hacemos un for para recorrer el objeto que nos toque… como no sabemos que objeto será, usamos --this—para referirnos a el.

Dentro del for al valor de arranque le damos this.lenth -1, es decir que arranque por el ultimo carácter del string… le damos que dure hasta que i sea igual o mayor a 0, asi dura hasta que terminamos el strin de atrás para adelante, y en la ultima indicación i--, es decir le decimos que desde el ultimo carácter vaya bajando hasta el primero.

Dentro del for le vamos sumando a la variable string que creamos para acumular los caracteres al revés la funcion this.Charat que nos retorna en cada índice del string la letra correspondiente. Finalmente devolvemos la variable creada.

Esta es otra forma de hacer la funcion sin usar el for

  String.prototype.reverse = function() {

   var stringinvertida = this.split('').reverse().join('');

  return stringinvertida;

}

}

También podríamos hacerlo asi: de este modo hhago un for normal y voy concatenando de derecha a izquierda lass letras de la iteración. Porque pongo esas variables a la izquierda del acumulador

String.prototype.reverse = function() {

   var stringinvertida = '';

   for (i = 0; i < this.length; ; i++) {

stringinvertida = this[i] + stringinvertida;

   }

return stringinvertida

  }

}

Usamos .this porque hacemos referencia al objeto que en un futuro crearemos a partir de la clase Persona.

Cuando armamos el constructor usamos this.propiedad, para armar cada propiedad, porqueasi decimos que la propiedad del objeto que vamos a crear será igual a lo que especifiquemos en los argumentos cuando lo hagamos

class Persona {

    constructor(nombre, apellido, edad, domicilio) {

    this.nombre = nombre,

    this.apellido = apellido,

    this.edad = edad,

    this.domicilio = domicilio,

    this.detalle = function() {

    return {

    Nombre : this.nombre,

    Apellido : this.apellido,

    Edad : this.edad,

    Domicilio : this.domicilio

    }

    }

    }

  }

También podría hacerse asi usando el método object.assign que copia el objeto this a la variable que nosotros le indiquemos

class Persona {

    constructor(nombre, apellido, edad, domicilio) {

    this.nombre = nombre,

    this.apellido = apellido,

    this.edad = edad,

    this.domicilio = domicilio,

    this.detalle = function() {

    var newobject= object.assign(newobject, this);

    return newobject

    }

    }

  }

function agregarMetodo() {

  //La función agrega un método "datos" a la clase Persona que toma el nombre y la edad de la persona y devuelve:

  //Ej: "Juan, 22 años"

  Persona.prototype.datos = function() {

    return this.nombre+ ', ' + this.edad +' años';

  }

  }

Podemos retornar asi con esta forma que se llama vastic (¿)

return `${this.nombre}, ${this.edad} años`;

` se llama acento grave. Se HACE CON ALT 96.

Esta forma de retornar el string hace que escribamos un string entero, encerrado entre `) y en el medio podamos agregar las variables antecedidas por un signo $

Definir métodos

La práctica común con funciones constructoras consiste en asignar métodos directamente a prototype en lugar de hacerlo con la initialización, como se aprecia en el método greet() a continuación.

constructor.js

function Hero(name, level) {

this.name = name;

this.level = level;

}

// Adding a method to the constructor

Hero.prototype.greet = function() {

return `${this.name} says hello.`;

}

Copy

Con clases, esta sintaxis se simplifica y el método puede agregarse directamente a las clases. Al usar la [abreviatura de definición de método](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Method_definitions) introducida en ES6, se logra una concisión aun mayor al definir un método.

class.js

class Hero {

constructor(name, level) {

this.name = name;

this.level = level;

}

// Adding a method to the constructor

greet() {

return `${this.name} says hello.`;

}

}