<https://vimeo.com/569374515/362e7cfe7b>

function crearGato(nombre, edad) {

  // Crear un nuevo objeto con la propiedad "nombre" y el valor definido como el argumento "nombre".

  // Agrega una propiedad al objeto con el nombre "edad" y usa el valor definido en el argumento "edad"

  // Agrega un método (funcion) llamado "meow" que devuelva el string "Meow!"

  // Devuelve el objeto

  // Tu código:

var Gato = {

  nombre : (nombre),

  edad : (edad),

  meow : function() {

    return ('Meow!');

  }

}

return Gato;

}

También puede ser así para no redundar en nombre y edad, ya que vienen definidos asi por argumento

var Gato = {

  nombre,

  edad,

  meow : function() {

    return ('Meow!');

  }

}

return Gato;

}

function agregarPropiedad(objeto, property) {

  // Agrega una propiedad al objeto (argumento "objeto") con el valor `null`

  // Devuelve el objeto

  // NOTA: El nombre de la propiedad no es "propiedad", el nombre es el valor del argumento llamado "property" (una cadena/string)

  // Tu código:

objeto[property] = null;

return objeto

}

Se usa bracket notation, porque te permite acceder a la propiedad sin saber el nombre de ella. Ya que no sabemos el nombre de la propiedad, sino que sabemos que el nombre de la propiedad es el valor del argumento property

function invocarMetodo(objeto, metodo) {

  // "metodo" es una cadena que contiene el nombre de un método (funcion) en el objeto

  // Invoca ese método

  // Nada necesita ser devuelto ("returned")

  // Tu código:

 return objeto[metodo]();

}

// No anda el objeto.property = null porque en el ej aclara que ‘metodo’ es una cadena que contiene el nombre de un método. Si no se el nombre de la propiedad no puedo usar dot notation. Invoamos el método, que es un afuncion con ()

function eliminarPropiedad(objeto, unaPropiedad) {

  // Elimina la propiedad de objeto cuyo nombre está pasado por el parametro unaPropiedad

  // tip: tenes que usar bracket notation

  // Devuelve el objeto

  // Tu código:

 delete objeto[unaPropiedad];

 return objeto;

}

Hay que usar bracket notation porque no conocempos el nombre de la propiedad, solo el argumento en el cual se pasa ese nombre

function multiplicarNumeroDesconocidoPorCinco(objetoMisterioso) {

  // "objetoMisterioso" tiene una propiedad llamada "numeroMisterioso"

  // Multiplica el numeroMisterioso por 5 y devuelve el producto

  // Tu código: var producto = (objetoMisterioso.numeroMisterioso \* 5); return producto;

//return (objetoMisterioso.numeroMisterioso \* 5);

return (objetoMisterioso['numeroMisterioso'] \* 5);

}

Lo podemos escribir con dot o con bracketnotation… en bracket notation debemos poner que es string porque conocemos el nombre de la propiedad

function tieneEmail(usuario) {

  // Devuelve "true" si el usuario tiene un valor definido para la propiedad "email"

  // De lo contratio, devuelve "false"

  // Tu código:

  if (usuario.email === null ||  usuario.email === undefined) {

  return false;

  }

  else {

    return true;

}

}

No hace falta compararlo con null o undefined en realidad. Con solo poner if(usuario.email) querria decir que es true, osea que no es ni null ni udefined entocnes es mejor preguntarlo asi.

if (usuario.email) {

return true;

}

return false;

}

No hace falta ponerle else al false, lo dejamos como default si no se cumple la condición….

También se puede resolver:

 if (usuario['email']){

   return true;}

   else return false

  }

Usamos las comillas en ‘email’ para asegurarnos que este definido es decir que sea null ni undefined

function tienePropiedad(objeto, propiedad) {

  // "propiedad" es un string

  // De lo contrario, devuelve "false"

  // Tu código:

if (objeto.hasOwnProperty(propiedad)) {

return true;

}

return false

}

También se puede hacer:

function verificarPassword(usuario, password) {

// Comprueba si la "password" enviada coincide con la propiedad "password" del objeto "usuario"

// Devuelve "true" si coinciden

// De lo contrario, devuelve "false"

// // Tu código:

  if (usuario.password === password) {

    return true;

  }

     return false;

  }

Como conozco el nombre de la propiedad uso do notation. invoco la propiedad del objeto en cuestión y la comparo con la password del argumento

function actualizarPassword(usuario, nuevaPassword) {

  // Reemplaza la contraseña existente en el objeto "usuario" con el valor de "nuevaPassword"

  // Devuelve el objeto

  // Tu código:

  usuario.password = nuevaPassword;

  return usuario;

}

También puede ser asi con []notation

usuario['password'] = nuevaPassword;

return usuario;

}

…Esta funcion hace referencia a la funcion anterior. Por eso no hacen referencia a la propiedad password del objeto usuario. Por eso yo conozco que el nombre es password…

function agregarAmigo(usuario, nuevoAmigo) {

  // "usuario" tiene una propiedad llamada "amigos" que es un array

  // Agrega "nuevoAmigo" al final de ese array

  // Devuelve el objeto "usuario"

  // // Tu código: tmb usuario.amigos.push(nuevoAmigo);

  usuario['amigos'].push(nuevoAmigo);

  return usuario;

}

También se puede hacer con dot notation

usuario.amigos.push(nuevoAmigo);

return usuario;

}

function pasarUsuarioAPremium(usuarios) {

  // "usuarios" es un array de objetos "usuario"

  // Cada objeto "usuario" tiene la propiedad "esPremium"

  // Define cada propiedad "esPremium" de cada objeto como "true"

  // Devuelve el array de usuarios

  // Tu código:

  // tmb usuarios[i].esPremium = true;

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

   usuarios[i].esPremium = true;

}

  return usuarios;

}

Se puede hacer con dot y bracket notation

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

usuarios[i]['esPremium'] = true;

}

return usuarios;

}

Podría usar for in y do while ttamben

function sumarLikesDeUsuario(usuario) {

  // "usuario" tiene una propiedad llamada "posts" que es un array

  // "posts" es un array de objetos "post"

  // Cada objeto "post" tiene una propiedad llamada "likes" que es un entero (int/integer)

  // Suma todos los likes de todos los objetos "post"

  // Devuelve la suma

  // Tu código:

  var suma = 0;

for (let i = 0; i < usuario.posts.length; i++) {

  suma = suma + usuario.posts[i].likes;

}

return suma;

}

Creamos la variante sums = 0 para ir sumando ahí todos los post.

A la suma le sumamos lo que esta en el objeto usuario, cuya propiedad posts es un array con mas objetos… iteramos el array posts y sumamos lo que esta en la propiedad likes de esos objetos.

Con br notation:

var suma = 0;

for (let i = 0; i < usuario.posts.length; i++) {

  suma = suma + usuario['posts'][i]['likes'];

}

return suma;

}

function agregarMetodoCalculoDescuento(producto) {

  // Agregar un método (función) al objeto "producto" llamado "calcularPrecioDescuento"

  // Este método debe multiplicar el "precio" del "producto" ("producto.precio" o "producto[precio]") y "porcentajeDeDescuento" para obtener el descuento

  // El método resta el descuento del precio y devuelve el precio con descuento

  // Devuelve el objeto "producto" al final de la función

  // Ejemplo:

  // producto.precio -> 20

  // producto.porcentajeDeDescuento -> 0.2 (o simplemente ".2")

  // producto.calcularPrecioDescuento() -> 20 - (20 \* 0.2)

  // Tu código:

  producto.calcularPrecioDescuento = function() {

 return (this.precio - (this.precio \* this.porcentajeDeDescuento));

     }

return producto;

}

Otra forma de hacerlo

producto.calcularPrecioDescuento = function () {

  var descuento = this.precio \* this.porcentajeDeDescuento

  var preciocondescuento = this.precio - descuento;

  return preciocondescuento;

}

return producto;

}

Es muy importante saber que this.precio y this. Porcentaje… están haciendo referencia a propiedades que tiene el objeto con el que trabajamos, es decir producto.

Otra forma de hacerlo es reemplazar los this por el nombre del objeto producto.calcularPrecioDescuento = function () {

  var descuento = producto.precio \* producto.porcentajeDeDescuento

  var preciocondescuento = producto.precio - descuento;

  return preciocondescuento;

}

return producto;

}

El .this va adentro de las propiedades y hace referencia al objeto en cuestión….. pero podríamos hacerlo también con los string y los array porque las clases ya creadas como array string etc. son objetos en si mismos

This. Nos sirve para referenciar al objeto en un método de un objeto…. Si lo hacemos en una funcion fuera del metodo referenciaeremos al objeto global