

# Es el momento de cosificar

*"Lo importante no es escuchar lo que se dice, sino averiguar qué es lo que se piensa."*

Juan Donoso Cortés – Escritor

En el encuentro aprendiste a crear las funcionalidades básicas para administrar una lista de tareas. Esta lista consiste de un array de strings, donde cada uno representa algo pendiente por hacer.

...

```
let tareas = [  
  "Terminar de leer la bitácora 21",  
  "Hacer el challenge",  
  "Empezar a ver The OA en Netflix",  
  "Aprender sobre arrays"  
];  
...
```

Sin embargo, una tarea es más que un título o descripción, ¿cierto? Si bien el enunciado de la tarea es de suma importancia ya que indica lo que tenemos que hacer, una tarea también pudiera tener información extra. Por ejemplo, ¿cuándo la registramos? ¿la hicimos ya? ¿cuándo la hicimos?.

El array nos permite crear listas de tareas con datos atómicos, como strings, pero, ¿cómo podemos llevar esta tarea a un modelo informativo con más datos? La respuesta: los objetos.

## Objetos

El objeto es una colección arbitraria de propiedades y valores. Un objeto en Javascript tiene esta forma:



...

```
let persona = {  
  nombre: "Manuel",  
  edad: 34,  
  ocupación: "Cantante",  
  soltero: true  
}
```

...

Una forma de crear objetos es declarando una variable cuyo valor es un bloque creado por llaves, en la que se podrán declarar duplas de propiedades y valores. El nombre de la propiedad puede ser escrito con o sin comillas. Lo siguiente representa una sintaxis válida de escribir un objeto:

...

```
let cafe = {  
  azucar: true,  
  "tamaño": 300  
}
```

...

## Propiedades

Los nombres de las propiedades comparten las limitaciones que estudiamos al principio del sprint sobre la declaración de nombres de variables como ``let`` y ``const``. Podemos escribir nombres de propiedades con o sin comillas. La ventaja de usar comillas para indicar el nombre de la propiedad es que podemos usar espacios en blancos o caracteres especiales como la "ñ".

Cuando seamos nosotrxs los que estemos creando los objetos, la descripción de la propiedad debe ser un nombre significativo para su valor. No es lo mismo escribir:



...

dato: 34

...

Que escribir:

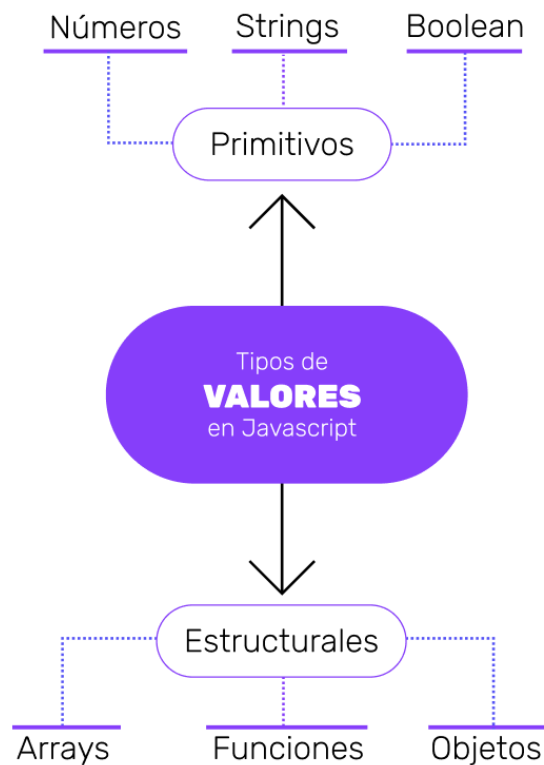
...

edad: 34

...

## Valores

El valor de la propiedad puede ser cualquier tipo de valor en Javascript, como los de este diagrama:



Esto quiere decir que podemos crear objetos con estructuras complejas, por ejemplo:

...



```
let ross = {
  amigxs: ["phoebe", "rachel", "joey", "chandler"],
  hermanxs: ["monica"],
  descendencia: true,
  casado: false,
  trabajo: "Paleontólogo"
}
...
```

Para mayor legibilidad designamos una línea entera a la dupla de la propiedad/valor, y las separamos con coma `,` en vez de punto y coma `;`.

## Leer valores

Para poder leer el valor de una propiedad dentro de un objeto, tenemos dos formas principales: la [notación de punto](#), y de [corchete](#).

La notación de punto consiste en escribir el nombre del objeto, un punto y el nombre de la propiedad que queramos leer.

```
...
let auto1 = {
  marca: "Toyota",
  modelo: "Yaris"
  color: "rojo",
  manual: false
};
console.log(auto1.modelo) // "Yaris"
...
```

La notación de corchetes (o *brackets notation* en inglés) es similar a la que utilizamos para acceder al index de un array, con la diferencia de que no



colocamos el número de posición, sino el nombre de una propiedad. Además, tenemos que colocar el nombre de la propiedad como un string.

```
...  
  
let encuentro22= {  
  nombre: "Objetos",  
  carrera: "DWFE"  
  visto: false,  
};  
  
console.log(encuentro22["nombre"]) // "Objetos"  
...
```

## Modificar valores

Con lo anterior, no solo estamos leyendo valores, sino que estamos directamente accediendo a ellos. Esto significa que podemos utilizar la notación de punto -o de corchete- para sobre escribir valores de propiedades.

```
...  
  
let encuentro22= {  
  nombre: "Objetos",  
  carrera: "DWFE"  
  visto: false,  
};  
  
console.log(encuentro22["visto"]); // false  
encuentro22["visto"] = true;  
console.log(encuentro22["visto"]); // true  
...
```



## Borrar propiedades

Con el operador ``delete`` podemos borrar propiedades de un objeto. Observa el siguiente código para ver su uso.

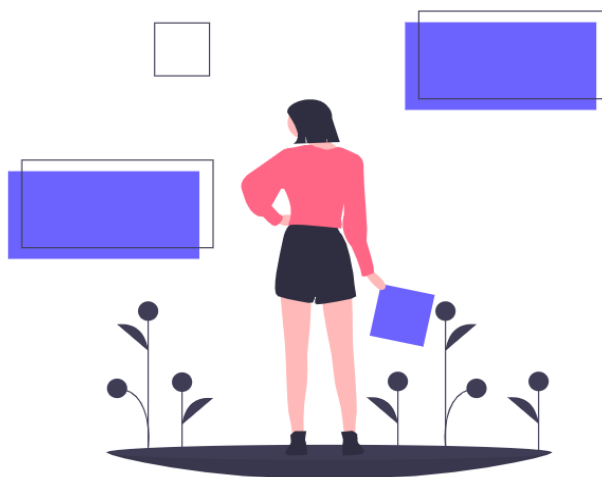
...

```
let plato1 = {  
  nombre: "hamburguesa",  
  vegetariano: false,  
  tomate: true  
}  
  
delete plato1.tomate; // también funcionaria delete plato1["tomate"]
```

...

## Objetificando

El objeto es una estructura de datos con duplas de propiedades y valores. La programación orientada a objetos nos permiten modelar de forma virtual algún entorno, cosa, persona o escenario de la vida real, literalmente todo puede ser un objeto. La clave para escribir objetos consiste en identificar "tipos" de objetos para así crear un molde, y extenderlo.



Por ejemplo, ¿cómo podrías *objetificar* los encuentros en zoom con tu equipo de aprendizaje? ¿Cuántos tipos de objeto puedes identificar?



Aquí va nuestra propuesta, podemos crear un objeto que represente a un estudiante, de esta forma:

...

```
let estudiante1 = {  
  nombre: "Fran",  
  trabajo: "Psiquiatra"  
};  
  
let estudiante2 = {  
  nombre: "Guido",  
  trabajo: "Soporte técnico"  
}  
...
```

Y también podemos crear un objeto para el equipo docente, por ejemplo:

...

```
let mentor = {  
  nombre: "Javi",  
  nacionalidad: "Venezolana"  
};  
  
let coach = {  
  nombre: "Debi",  
  nacionalidad: "Argentina"  
}  
...
```

Pero, identificar tipos de objeto consiste en un ejercicio de abstracción y comparación, en el que podemos también ver propiedades en común. ¿Qué tienen en común un/a coach, un estudiante y un/a squad lead? Son personas.



Por lo tanto, podemos tener un objeto `persona` que tenga tanto las propiedades en común, como el diferencial:

...

```
let persona1 = {  
  nombre: "Guido",  
  tipo: "estudiante",  
  trabajo: "Soporte técnico"  
};
```

```
let persona 2 = {  
  nombre: "Javi",  
  tipo: "mentor",  
};
```

```
let persona3 = {  
  nombre: "Debi",  
  tipo: "coach",  
  nacionalidad: "Argentina"  
};
```

...

Un conjunto de objetos que estén relacionados bajo algún criterio lógico, no necesariamente tienen que tener la misma estructura. En el ejemplo anterior se muestran tres objetos `persona` pero que solo tienen una propiedad en común, el `tipo` y `nombre`.

## Propiedad `this`

Como te mencionamos antes, el objeto es una estructura compuesta por una dupla de una propiedad con un valor:





```
let tarea = {  
  descripcion: "leer la bitácora",  
  completada: false  
}
```



propiedad      valor

Para el valor, se puede especificar cualquiera de los disponibles vistos hasta ahora, primitivos o estructurales. Esto significa que el valor de una propiedad de un objeto, puede ser otro objeto, un array, o inclusive una función.

En Javascript, todos los objetos tienen disponible la propiedad `this``. Esta es una propiedad especial que se utiliza para acceder a valores del mismo objeto. Observa el siguiente video donde mostramos como un valor de una propiedad puede ser una función, junto con un uso típico de la palabra reservada `this``.

[Propiedad this](#)

## En resumen

El objeto es una estructura de datos muy flexible dentro de los tipos de valores de Javascript. Es una colección desordenada de duplas de propiedades y valores. A diferencia del array, que se caracteriza por tener una secuencia numerada, el objeto no presenta ningún tipo de orden.


Los valores de las propiedades de un objeto pueden ser cualquiera de los tipos de valores que ofrece Javascript como lenguaje, inclusive, otros objetos. La propiedad especial `this`` nos permite auto referenciar propiedades de un objeto.

Mientras nos seguimos adentrando en el mundo de la programación, con la compañía de Javascript, vamos a notar como el objeto se convierte en la estructura de datos de preferencia para crear modelos de datos más complejos, y a la vez, más flexibles.





## ¡Prepárate para la próxima meeting!

### Challenge

 1. Crea un objeto que tenga propiedades de una persona. Como mínimo el nombre, y la edad. Crea una propiedad dentro de este objeto cuyo valor sea una función anónima que determine si la persona es mayor de edad o no. Considera la mayoría de edad como un valor mayor o igual a 18. Esta función no debe retornar nada, solo mostrar por consola el mensaje respectivo. Por ejemplo:

```
let persona = {  
  nombre: "Susana",  
  edad: 20,  
  esMayorDeEdad: ...  
}  
persona.esMayorDeEdad();  
// Susana es mayor de edad.
```

 2. Partiendo de [este sandbox](#) completa la función "disponible" la cual indica si el producto está disponible o no. Un producto no está disponible cuando el valor de "cantidad" es menor o igual a cero. Mostrar los mensajes correspondientes para cada caso. Sobre el mismo sandbox, modifica la propiedad "cantidad" para que tenga un valor de cero, y vuelve a invocar la función "disponible".

 3. Declara un array de objetos. Diseña cada objeto considerando que cada uno debe representar una tarea dentro del Taskineitor. Tienes la libertad de crear las propiedades que consideres, como título o nombre de la tarea, el estado en el que se encuentra, (completada/no completada), entre otras. Trae tu array de tareas para la próxima clase.