



# Objetos definidos por el usuario

Desarrollo Web en Entorno Cliente

Ciclo Superior de Desarrollo de Aplicaciones Web

2020/2021

# Tipos de objetos

- Los objetos nativos son aquellos proporcionados por JS. Ejemplos de objetos nativos son String, Number, Array, Date, Math, etc.
- Los objetos definidos por el usuario son aquellos creados por el programador.



# Objetos definidos por el usuario

- Los objetos en JS son similares a los *arrays*, pero los objetos emplean cadenas en lugar de índices numéricos para acceder a los elementos que almacenan.
- Estas cadenas se llaman claves o propiedades; los elementos a los que apuntan se denominan valores. El conjunto se denomina pares clave-valor.
- Mientras que los *arrays* se emplean mayormente para representar listas de elementos, los objetos suelen emplearse para representar una única entidad con varias características o atributos.



# Creación de objetos

- Para crear un objeto se emplean las llaves { }, en lugar de los corchetes [ ] empleados para crear *arrays*.
- Entre las llaves se establecen los pares clave-valor.
- En conjunto, las llaves y los pares clave-valor se denominan literal de objeto.
- Un literal de objeto es un modo de crear un objeto escribiendo el objeto completo.



# Creación de objetos

```
// creación de un objeto  
let persona = {  
  nombre: "Juan",  
  apellido: "Pérez"  
};
```



# Creación de objetos

- Las claves siempre van a ser cadenas, motivo por el cual en algunos casos las comillas pueden obviarse en el literal del objeto.
- En el caso de no emplear comillas para las claves, éstas deben seguir las mismas reglas que los nombres de variable: por ejemplo, no se permiten los espacios. Si una clave necesita espacios, debe establecerse entre comillas.



# Creación de objetos

```
// creación de un objeto  
let persona = {  
  nombre: "Juan",  
  "Primer apellido": "Pérez",  
  "Segundo apellido": "López",  
};
```



# Acceso a los valores de objetos

- Para acceder a los valores de un objeto se emplean los corchetes [ ], del mismo que se emplean en los *arrays*.
- La diferencia es que en lugar de emplear índices (un número), se emplean claves (una cadena).

```
let miNombre = persona["nombre"];
```

- Si se solicita una clave que no existe, JS devuelve un valor `undefined`.





# Acceso a los valores de objetos

- Del mismo modo que las comillas son opcionales cuando se crea el literal del objeto, también son opcionales cuando se accede al valor, pero si no se emplean comillas, la sintaxis cambia.
- Esto es lo que se llama notación de punto. Se emplea un punto seguido de la clave, sin comillas, en lugar de introducir la clave entre corchetes y comillas.

```
let miNombre = persona.nombre;
```

- Del mismo modo que ocurría en los literales de objeto, esta notación es válida únicamente si la clave no contiene caracteres especiales, como los espacios.

```
let miNombre = persona["Primer apellido"];
```



# Recuperar las claves de un objeto

- Puesto que un objeto es una colección de pares clave-valor, es útil conocer todas las claves de un objeto.
- `Object.keys(unObjeto)` devuelve un *array* con los valores de todas las claves del objeto que recibe `unObjeto`.



# Añadir valores

- JavaScript es flexible en cuanto al acceso y modificación de las propiedades de un objeto.
- Los objetos pueden ser creados dinámicamente.
- Del mismo modo que se accede a las propiedades de un objeto, se pueden añadir nuevas propiedades.
- Cuando se da valor a una propiedad que no existe, se crea dicha propiedad.



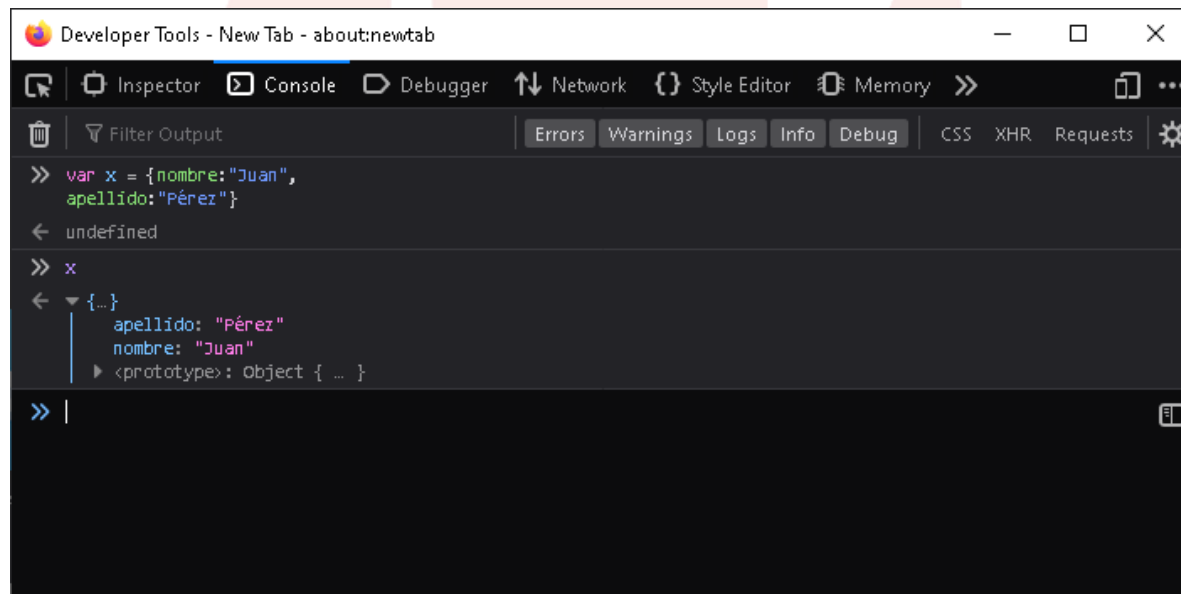
# Añadir claves y valores a objetos

```
// creación de un objeto:  
let persona = {  
  nombre: "Juan",  
  "Primer apellido": "Pérez",  
  "Segundo apellido": "López",  
};  
// creación de una nueva propiedad  
edad:  
persona.edad = 30;
```



# Inspección de objetos en la consola

- La consola de las herramientas de desarrollador de los navegadores permiten inspeccionar el contenido de los objetos.



The screenshot shows the 'Developer Tools' window of a browser, specifically the 'Console' tab. The console displays the following code and output:

```
>> var x = {nombre:"Juan",  
          apellido:"Pérez"}  
← undefined  
  
>> x  
← {...}  
  apellido: "Pérez"  
  nombre: "Juan"  
  > <prototype>: Object { ... }
```

The object `x` is expanded to show its properties: `apellido` with value `"Pérez"` and `nombre` with value `"Juan"`. The `<prototype>` property points to `Object { ... }`.



# Añadir métodos

- En JS una variable puede almacenar una función.
- En el caso de los objetos, las propiedades también pueden tener como valor una función.
- Un método es una función almacenada en una propiedad de un objeto.



# Palabra clave `this`

- Se puede emplear la palabra clave `this` dentro de un método para referirse al objeto en el cual el método está siendo invocado.
- `this` hace que los métodos sean más versátiles, permitiendo añadir el mismo método a varios objetos y tener acceso a las propiedades concretas del objeto sobre el cual se ejecuta.



# Constructores

- Para compartir métodos y propiedades entre objetos, se pueden añadir todos ellos a cada uno de los objetos.
- Pero si se maneja un número elevado de objetos, propiedades o métodos, añadirlos de modo individual se vuelve tedioso y complica el código.
- Los constructores ofrecen un modo mejor de compartir métodos y propiedades entre objetos.





# Constructores

- En JS un constructor es una función que crea objetos con una serie de propiedades y métodos predefinidas.
- Si un objeto cuenta con un constructor, éste se puede emplear para construir objetos fácilmente.
- Cada vez que se llama a un constructor, éste crea un nuevo objeto, incluyendo propiedades y métodos.



# Añadir claves y valores a objetos

```
// definición de constructor
function Persona (nom, ape){
  this.nombre = nom;
  this.apellido = ape;
}
// llamada al constructor
let persona = new Persona ("Juan", "Pérez");
```



# Constructores

- Para llamar a una función normal, se emplea su nombre seguido de un par de paréntesis.
- Para llamar a un constructor, se emplea la palabra clave new, seguida del nombre del constructor y los paréntesis.



# Prototipos

- JS es un lenguaje basado en prototipos.
- La programación basada en prototipos es un tipo de programación orientada a objetos en la que no existe el concepto de clase.
- En lugar de a través de clases, la reutilización de comportamiento (denominada herencia en programación basada en clases) se consigue clonando objetos ya existentes denominados **prototipos**.



# Prototipos

- Todos los constructores de JS tienen una propiedad `prototype` a la que se pueden añadir métodos y propiedades.
- Cualquier método o propiedad que se añada a un prototipo estará disponible para todos los objetos creados mediante el constructor al que pertenece el prototipo.



# Añadir métodos a través de prototipos

```
luis.edad = "20";
```

```
/* Se crea la propiedad edad  
solamente para el objeto luis */
```

```
Persona.prototype.edad = "20";
```

```
/* Todos los objetos que se creen  
mediante el constructor Persona  
tendrán la propiedad edad */
```



# Referencias

- [JavaScript en w3schools](#)
- [JavaScript en Mozilla Developer Network](#)





FORMACIÓN PROFESIONAL  
MONTECASTELO