

PRÁCTICA 2: OBJETOS JAVASCRIPT

1. INTRODUCCIÓN

Desde la norma de **ECMAScript5** Javascript puede considerarse un lenguaje de programación orientado a objetos, pero con algunas peculiaridades.

JavaScript es un lenguaje de programación orientado a objetos pero basado en prototipos.

Javascript no tiene clases solo definiciones de objetos, constructores, que se usan como modelos para construir otros objetos. Estos objetos padre o modelos serían lo más parecido a las clases de la programación orientada a objetos.

Aunque en la última versión aparezca una estructura semántica con el nombre **class**. Estas clases son en realidad constructores tipo **Function**, pero aportan algunas facilidades como **extends** y métodos estáticos (**static**).

De hecho Javascript trabaja con un objeto global que gobierna a todos los demás objetos (es el señor de los objetos).

Las clases son "funciones especiales", como las **expresiones de funciones** y **declaraciones de funciones**, la sintaxis de una clase tiene dos componentes: **expresiones de clases** y **declaraciones de clases**.

- **Declaración de clases:** Una manera de definir una clase es mediante una declaración de clase. Para declarar una clase, se utiliza la palabra reservada **class** y un nombre para la clase "Rectangulo".

```
class Rectangulo {  
  constructor(alto, ancho) {  
    this.alto = alto;  
    this.ancho = ancho;  
  }  
}
```

Una importante diferencia entre las declaraciones de funciones y las declaraciones de clases es que las declaraciones de funciones son alojadas y las declaraciones de clases no lo son. En primer lugar necesitas declarar tu clase y luego acceder a ella, de otro modo el ejemplo de código siguiente arrojará un **ReferenceError**:

- **Expresión de clase:** Una expresión de clase es otra manera de definir una clase. Las expresiones de clase pueden ser nombradas o anónimas. El nombre dado a la expresión de clase nombrada es local dentro del cuerpo de la misma.

```
// Anonima  
let Rectangulo = class {  
  constructor(alto, ancho) {  
    this.alto = alto;  
    this.ancho = ancho;  
  }  
}
```

```
};

console.log(Rectangulo.name);
// output: "Rectangulo"

// Nombrada
let Rectangulo = class Rectangulo2 {
  constructor(alto, ancho) {
    this.alto = alto;
    this.ancho = ancho;
  }
};
console.log(Rectangulo.name);
// output: "Rectangulo2"
```

- **Constructor:** El método constructor es un método especial para crear e inicializar un objeto creado con una clase. Solo puede haber un método especial con el nombre "constructor" en una clase. Si esta contiene mas de una ocurrencia del método constructor, se arrojará un Error SyntaxError

2. HERENCIA

La herencia significa que cuando se crea un objeto a partir de otro (el constructor), este objeto puede acceder a las propiedades y métodos de la clase que lo ha creado.

¿Como hace esto Javascript? Con prototype. Hay quien dice que más que basado en objetos, Javascript es un lenguaje basado en prototipos.

El objeto **prototype** no es más que eso: un objeto básico de un constructor que contiene todos los métodos y propiedades que serán accesibles a los objetos basados en ese constructor.

¿Como funciona este objeto? Si desde un objeto se invoca un método o una propiedad el intérprete de javascript busca entre sus métodos o propiedades. Si no lo encuentra, busca en el **prototype** del objeto constructor. Si está se ejecuta. Si tampoco está repite el proceso: va al constructor de este objeto y busca en su prototype. Y así hasta encontrarlo o emitir un error si no existe en el último constructor. Eso es se suele conocer como la cadena prototype.

La palabra clave **super** es usada para llamar funciones del objeto padre.

Ejemplo:

```
class Gato {
  constructor(nombre) {
    this.nombre = nombre;
  }

  hablar() {
    console.log(this.nombre + ' hace ruido.');
```

```
class Leon extends Gato {
  hablar() {
    super.hablar();
    console.log(this.nombre + ' maulla.');
```

3. ¿Qué es el local Storage?

Cuando construimos una app, normalmente trabajamos con algún tipo de información, por ejemplo, si le pedimos a un usuario que introduzca una ciudad para ver qué tiempo hace allí. Pero si no almacenamos esa información en algún sitio, ésta se perderá cuando recarguemos la página.

Una de las maneras de solucionarlo es utilizando **local storage**: La información queda almacenada en el navegador del usuario. Esa información, al igual que en una base de datos, podremos rescatarla y manipularla como queramos. La **local storage API** nos la proporciona el navegador directamente, así que podemos acceder a esa a través del **window object**. Si vamos a la consola y escribimos `window`, verás que una de las propiedades es **localStorage**. Dentro de ella verás que hay una propiedad llamada `length`, que representa el número de elementos que tenemos almacenados en `localStorage` actualmente.

Hemos dicho entonces que en `localStorage` almacenamos datos. Cada dato será un item que tendrá una **key** y un **value**, muy parecido a la estructura de un objeto en JavaScript (JS). Con la particularidad de que un item debe ser siempre un string. Eso no significa que no podamos crear arrays, objetos, o cualquier otro data type. Simplemente significa que cualquier data type debe ir entre comillas como un **string** para poder ser almacenado en **local storage**.

Por lo tanto para almacenar datos en el `localStorage` lo que haremos es convertir los objetos de tipo JavaScript en objetos JSON

- **Cómo almacenar y obtener datos**
 - Para almacenar datos en local storage, usamos el método `setItem`.
Sintaxis: **`localStorage.setItem('key name', 'value name');`**
 - Para obtener datos almacenados en local storage, utilizamos el método llamado **`getItem`** y le pasamos como argumento el key name del item que queremos obtener.
 - También podemos actualizar información, simplemente usamos el método **`setItem`** de nuevo y sobrescribimos los valores.
- **Cómo borrar datos:** Existen dos posibilidades: borrar un solo item de local storage o borrarlos todos.
 - Para borrar un solo item, usamos el método **`remove`** y le pasamos por parámetro el key name del item que queremos borrar.
 - Si lo que queremos es borrar todos los datos que tenemos en local storage, usamos el método **`clear()`**.
- **Cómo convertir datos a formato JSON / desde formato JSON:**

Hasta ahora hemos trabajado con datos sencillos, como strings y números. Es momento de trabajar con estructuras más complejas, como arrays u objetos Recuerda que los datos que

guardamos en local storage deben de tener el formato de string. Así que, guardemos lo que guardemos, la local storage API lo convertirá en un string,

Para guardar esta estructura compleja en local storage, primero debemos convertir esos datos a un JSON string. Para eso, usamos un método del JSON object llamado **stringify** y le pasamos la variable como argumento. 3

- `localStorage.setItem('toDosList', JSON.stringify(toDos));`

Los siguientes pasos serían obtener esos datos (retrieve) y convertirlos de nuevo en un array para poder manipularlo. Para ello, creamos una variable llamada `storedToDos`, y usamos el método `getItem` para obtener los datos. Esto nos seguiría dando un JSON string, así que usamos el método `parse` sobre los datos.

- `const storedToDos = localStorage.getItem('toDosList');`
- `console.log(JSON.parse(storedToDos))`

- **Buscar una clave en el localStorage:** Para ello debemos recorrer todas las claves del almacenamiento local para comprobar la existencia o no de una clave.


A veces debemos leer los datos que están almacenados en el `localStorage`, pero no conocemos las claves exactas. Podemos obtener el nombre de la clave utilizando el método **key()**. Dicho método acepta un índice como argumento y devuelve el nombre de la clave que coincide con el índice.

Podemos recorrer el `localStorage` con un bucle y leer los datos sin conocer las claves exactas

```
let nextValue;  
for (let i = 0; i < localStorage.length; i++){  
    nextValue = localStorage.getItem(localStorage.key(i));  
    //Do something with nextValue..  
    //..  
}
```

Secuencia y desarrollo:

1. **Ejercicio 1: Implementa la siguiente página y almacena los datos de los alumnos. Almacena los datos de forma local.**



The screenshot shows a web browser window with multiple tabs. The active tab is titled 'JSON' and shows a file path in the address bar: file:///G:/CURSO23_24/DWCC_DISTANCIA/CLASES/Practica2_LocalStorage/p2_ud4.html? . The page content includes a section titled 'Inserta Alumnos' with a form containing three input fields labeled 'Nombre', 'Dirección', and 'DNI', and two buttons labeled 'Guardar' and 'Mostrar'. Below this is a section titled 'Alumnos' with a label 'Eliminar alumnos' and a button labeled 'Borrar'.

PASOS:

1. Crea un objeto javascript ALUMNO con los siguientes campos
 - Nombre:
 - DNI:
 - Dirección
2. Añade los metodos:
 - Constructor
 - Modificar
 - Mostrar
3. Convierte dicho objeto a un objeto json para almacenar en el localStorage
4. Añade un botón Mostrar el cual recorre el localStorage y lo convierte en objeto JavaScript para mostrarlos en una tabla.

Inserta Alumnos

Alumnos

Nombre:
Dirección:
DNI:
Guardar Mostrar

Alumnos

DNI	NOMBRE	DIRECCION	
35252525X	Luis Perez Perez	Vigo	Modificar
12312456S	Ana Rodas Rodas	Coruña	Modificar

Eliminar alumnos: Borrar

Inspector Console Depurador Rede Editor de estilos Rendemento Memoria Almacenamento Accesibilidade Aplicativo

Almacenamiento local

key Value

http://127.0.0.1:5500

Almacenamiento de sesión

Almacenamiento en caché

Cookies

Indexed DB

Como muestra la imagen anterior, se introducen los datos de cada alumno en el formulario. Al pulsar Guardar se almacena en LocalStorage y al pulsar Mostrar lo muestra en la tabla.

5. Al pulsar modificar aparece los datos del alumno en el formulario y se pueden modificar

Inserta Alumnos

Alumnos

Nombre: Luis Perez Perez
Dirección: Vigo
DNI: 35252525X
Modificar Mostrar

Alumnos

DNI	NOMBRE	DIRECCION	
35252525X	Luis Perez Perez	Vigo	Modificar
12312456S	Ana Rodas Rodas	Coruña	Modificar

Eliminar alumnos: Borrar

Inspector Console Depurador Rede Editor de estilos Rendemento Memoria Almacenamento Accesibilidade Aplicativo

Almacenamiento local

key Value

http://127.0.0.1:5500

Almacenamiento de sesión

Almacenamiento en caché

Cookies

Indexed DB

En este ejemplo se modifica los datos del alumno Luis Perez, aunque el único campo que no se puede modificar es la clave que en este caso es el DNI, por lo que debe ser un campo no editable en el momento de realizar la modificación

Una vez que se cambia, en este caso la dirección, el resultado es el mostrado a continuación.

Curso: 36011 | Editar o curso: Practica5_JavaScri... Practica5_JavaScri... JSON JSON JSON P1_UD4.pdf Objetos en JavaScri... Document JSON JSON JSON x + - □ x

127.0.0.1:5500/p2_ud4.html?

Inserta Alumnos

Alumnos

Nombre
Dirección
DNI

Guardar Mostrar

Alumnos

DNI	NOMBRE	DIRECCION	
35252525X	Luis Perez Perez	Lugo	Modificar
12312456S	Ana Rodas Rodas	Coruña	Modificar

Eliminar alumnos

Borrar

Inspector Console Depurador Rede Editor de estilos Rendemento Memoria Almacenamento Accesibilidade Aplicativo

Filtrar os elementos

Almacenamento local

key

Value

Almacenamento da sesión

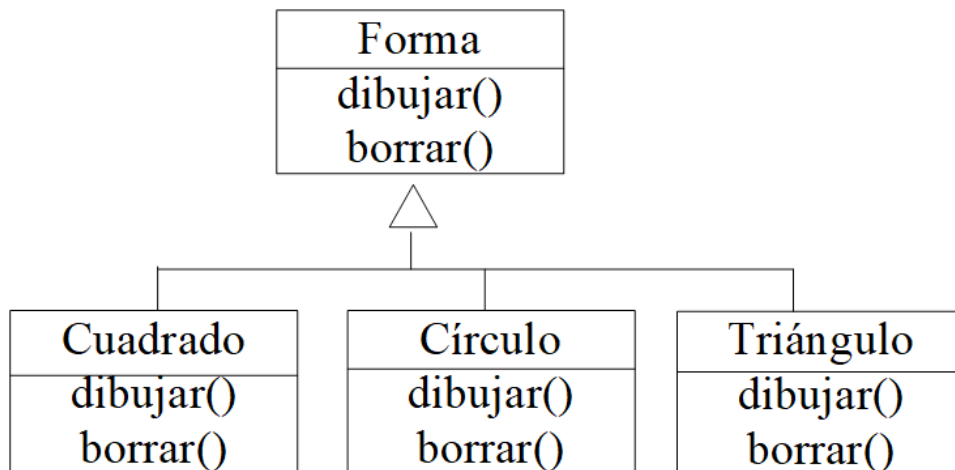
Almacenamento na caché

Cookies

Indexed DB

6. Para eliminar un alumno, debemos escribir su dni en el cuadro de eliminar alumno, se obtiene del localStorage por su key y se elimina.

2. Ejercicio 1: En este ejercicio vamos a trabajar con clases abstractas y clases derivadas. Para ello debes crear la siguiente estructura de clases.



Este es el código para crear la clase abstracta Forma.

```

class Forma {
  constructor() {
    if (this.constructor === Forma)
      throw new Error("La clase Forma es abstracto");
  }
}
    
```



```
}  
dibujar() {  
    throw new Error(  
        "Este es un método abstracto y se debe sobrecribir en las subclases"  
    );  
};  
}
```

A continuación debes implementar una página Web que permita al usuario elegir una de las tres figuras (**cuadrado**, **triángulo** y **círculo**).

Crear un objeto de ese tipo e implementar el método dibujar el cual indica la figura creada. Para mostrar el mensaje debes utilizar los métodos de la estructura DOM.