Unidad 2 - Objetos y tipos de datos

Objetos y propiedades

Crear un objeto literal con propiedades

```
let miObjeto = {
    prop1: val1,
    prop2: val2,
    "prop 3": val3,
}
```

- `prop1`, `prop2`: Son propiedades cuyo nombre debe ser un identificador de variable válido. Pueden ser palabras reservadas.
- `"prop 3"`: Si las propiedades están entre comillas, pueden tener cualquier nombre.

Crear un objeto vacío

```
let miObjeto = {};
let miObjeto = new Object();
```

Acceder a una propiedad usando el operador punto

```
miObjeto.prop1;
```

Acceder a una propiedad con corchetes `[]`

```
miObjeto["prop1"];
```

- `"prop1"`: Es una cadena de texto. Puede ser una variable o el resultado de una concatenación.

```
x = "prop2";
console.log("valor de " + x + ": " + miObjeto[x]);
```

Añadir una propiedad

```
miObjeto.prop4 = val4;
miObjeto["prop4"] = val4;
```

Eliminar una propiedad

```
delete miObjeto.prop4;
delete miObjeto["prop4"];
```

Atajo para nombres de propiedades

```
nombre = "Jose";
apellido = "Pérez";
persona = {nombre, apellido};

// {nombre: 'Jose', apellido: 'Pérez'}
```

Si una propiedad no existe

Si intentas acceder a una propiedad que no existe, el resultado será `undefined`.

Operador `in`

Permite comprobar si una propiedad existe en un objeto.

```
"prop" in objeto;
"nombre" in persona;
```

Bucle `for .. in`

```
for (let prop in persona) {
    // props
    alert(prop); // nombre, apellido
    // valores de las props
    alert(persona[prop]); // José, Pérez
}
```

Referencia

Objetos y valores por referencia

- Las variables de tipo objeto contienen la referencia (dirección de memoria) del lugar donde está almacenado el objeto.

  ```
  let persona2 = persona;
  ```

- Si se asigna un objeto a otra variable, ambas variables apuntarán al mismo objeto, no se crea una copia.
- Dos variables apuntan al mismo objeto si su comparación devuelve `true`. Si son dos objetos iguales pero diferentes, la comparación será `false`.

`Object.assign`

- Copia las propiedades de los objetos fuente a un objeto destino.
- Devuelve el objeto destino.

  ```
  Object.assign(dest, ori1, ori2, …);
  ```

Clonar un objeto

```
persona2 = Object.assign({}, persona);
```

Clonación profunda

- Se utiliza si el objeto contiene otros objetos anidados.

```
persona2 = structuredClone(persona);
```

Recolección de basura

- La gestión de memoria en JavaScript es automática. El motor se encarga de eliminar de la memoria los objetos que ya no son accesibles.

Métodos

Qué son

- Son propiedades cuyo valor es una función.

Declarar un método

Existen tres formas de declarar un método:

- Asignar una función anónima:

```
let perro = {
    ladrar : function () {
        alert("guau")
    },
}
```

- Asignar una función existente:

```
let perro = {
    ladrar : ladrarPerro,
}
```

- Usar la sintaxis simplificada:

```
let perro = {
    ladrar() {
        alert("guau")
    },
}
```

`this`

- `this` se puede utilizar en funciones.

- `this` se evalúa en tiempo de ejecución dependiendo del contexto:
  - Si la función se llama en el ámbito global, `this` hace referencia al objeto global (`window` en navegadores o `global` en Node.js).
  - Si la función se llama como un método de un objeto, `this` apunta a ese objeto.
  - Si la función se ejecuta usando `new`, `this` se refiere al nuevo objeto creado.
  - Las funciones flecha no tienen su propio `this`, lo heredan del contexto exterior. Por esto, no deben usarse para:
    - Crear métodos de objetos.
    - Callbacks con contexto dinámico.

Funciones constructoras, operador `new`

Funciones constructoras

- Son funciones normales usadas para crear objetos.
- No pueden ser funciones flecha.
- Se recomienda que comiencen con mayúscula por convención.
- No es común que tengan una sentencia `return`.

Cómo crear un objeto a partir de una función constructora:

- Usa el operador `new` seguido de la llamada a la función.

¿Qué ocurre al ejecutar una función con `new`?

- Se crea un objeto vacío y se asigna a `this`.
- Se ejecuta la función, modificando `this` y añadiendo propiedades.
- Se devuelve el valor de `this`.

```
function User(name) {
    // this = {}; (implícitamente)
    // agrega propiedades a this
    this.name = name;
    this.isAdmin = false;
    this.saluda = function () {
        alert("Hola, soy " + this.name);
    };
    // return this; (implícitamente)
}

let nuevoUser = new User("Pepe");

nuevoUser.saluda();
```

```
function Gato() {
    this.saluda = function () {
```

```
        console.log("Hola, soy un gato");
    };
}

let miGato = new Gato();
let otroGato = new Gato;

miGato.saluda();
otroGato.saluda();
```

- Si el constructor no necesita parámetros, se pueden omitir los paréntesis

Tipos de datos

Métodos de tipos primitivos

Los tipos primitivos como `string`, `number`, y `boolean` tienen métodos asociados, aunque `null` y `undefined` no.

Tipo `number`

Escribir números

Números pueden ser escritos con notación científica (1e6), separadores (_), o bases diferentes como hexadecimal (0xff).

```
let millon = 1000000;
let millon2 = 1_000_000; // Azúcar sintáctica
let millon3 = 1e6; // 1 * 10 elevado a 6
```

```
let valor = 255; // decimal
let valor2 = 0xff; // hexadecimal
let valor3 = 0xff; // hexadecimal
let valor4 = 0o377; // octal
let valor5 = 0b11111111; // binario
```

Uso de dos puntos para llamar a un método

- Necesario si aplicamos el método directamente a un número.

  ```
  console.log(1234..toString(2));
  ```

- Se puede evitar si utilizamos paréntesis.

  ```
  console.log((1234).toString(2));
  ```

Devuelve una cadena con el valor del número en una base determinada `num.toString([base])`

- `base` puede tomar valores de 2 a 36 (por defecto es 10)

```
// Muestra el valor en decimal (base por defecto)
console.log(valor.toString());
// Muestra el valor en base 2
console.log(valor.toString(2));
// Muestra el valor en base 16 (hexadecimal)
console.log(valor.toString(16));
```

Redondeo y decimales

Métodos como `Math.floor()`, `Math.ceil()`, `Math.round()`, y `Math.trunc()` para redondear.

`Math.floor( numero )` → Redondea hacia abajo

`Math.ceil( numero )` → Redondea hacia arriba

`Math.round( numero )` → Redondea al entero más próximo

`Math.trunc( numero )` → Trunca la parte decimal

```
console.log(Math.floor(3.1)); // 3
console.log(Math.ceil(3.1)); // 4
console.log(Math.round(3.1)); // 3
console.log(Math.trunc(3.1)); // 3
```

`toFixed()` para redondear y devolver un número con decimales.

```
valor = 12.3456;
console.log(num.toFixed(1)); // "12.3"
console.log(num.toFixed(2)); // "12.35"
```

Métodos booleanos con números

`isNaN()`, `isFinite()`, y sus equivalentes de `Number` verifican si un valor es válido o finito.

`isNaN(valor)` → Devuelve true si el valor es `NaN`

`isFinite(valor)` → Devuelve true si el valor no es `NaN`, `Infinity` o `-Infinity`

`Number.isNaN(valor)` → Devuelve true si el valor es `number` y es `NaN`

`Number.isFinite(valor)` → Devuelve `true` si el valor es `number` y no es `NaN`, `Infinity` o `-Infinity`

```
console.log(isFinite(15)); // true
console.log(isFinite("15")); // true
console.log(isFinite("asdf15")); // false

console.log(Number.isFinite(15)); // true
console.log(Number.isFinite("15")); // false
console.log(Number.isFinite("asdf15")); // false
```

Parsear cadenas

`parseInt()` y `parseFloat()` extraen números de cadenas.

- `parseInt(cadena)` → Devuelve un entero
- `parseFloat(cadena)` → Devuelve un punto flotante

```
console.log(parseInt("100px")); // 100
console.log(parseInt("1.5em")); // 1
console.log(parseFloat("100px")); // 100
console.log(parseFloat("1.5em")); // 1.5
console.log(parseInt("a 1.5em")); // NaN
```

Aleatorios

`Math.random()` genera números entre 0 y 1. Funciones adicionales permiten ajustar rangos y convertirlos en enteros.

- Función que devuelve un número de punto flotante entre min y max (pero no incluyendo max):

```
function random(min, max) {
    return min + Math.random() * (max - min);
}
```

- Función que devuelve un número entero entre min y max (incluyendo tanto min como max):

```
function randomInteger(min, max) {
    // aquí rand es desde min a (max+1)
    let rand = min + Math.random() * (max + 1 - min);
    return Math.floor(rand);
}
```

Otras funciones de `Math`

- Devuelve el valor máximo de los valores que se pasan como argumentos:

```
Math.max(v1, v2, ...);
```

- Devuelve el valor mínimo de los valores que se pasan como argumentos:

```
Math.min(v1, v2, ...);
```

- Devuelve v1 elevado a v2

```
Math.pow(v1, v2);
```

Tipo `string`

Caracteres especiales

Para incluir caracteres especiales en una cadena, se utiliza la barra invertida `\` para "escapar" caracteres como `\n` (nueva línea), `\'`, `\"`, `\`` y `\\` (barra invertida).

```
'¡Yo soy la \'morsa\'!'
```

Caracteres de un `string`

- `str.length` → Devuelve la longitud del `string`.
- Acceder a caracteres
  - `str[pos]` o `str.at(pos)` → El primer carácter está en la posición 0, y el último en `str.length - 1`.
  - Con `.at(pos)` puedes usar índices negativos: `str.at(-1)` devuelve el último carácter.

```
let str = `Hola`;

// el primer carácter
console.log( str[0] ); // H
console.log ( str.at(0) ); // H

// el último carácter
console.log ( str[str.length - 1] ); // a
console.log ( str.at(-1) );
```

- `toUpperCase()` y `toLowerCase()` → Convierte a mayúsculas o minúsculas.

```
let saludo = "Hola";
console.log(saludo.toUpperCase()); // HOLA
console.log(saludo.toLowerCase()); // hola
```

Búsqueda de subcadenas

- `str.indexOf(substr, [from])` → Devuelve la posición de `substr`, comenzando desde `from`. Si no la encuentra, devuelve -1.
- `str.lastIndexOf(substr, [from])` → Igual, pero busca de atrás hacia adelante.

```
let str = "Widget con id";
console.log(str.indexOf("id", 2)); // 11
console.log(str.lastIndexOf("id", 2)); // 1
```

Verificar presencia de una subcadena

- `str.includes(substr, [from])` → Devuelve `true` si `substr` está en `str`.
- `str.startsWith(substr, [from])` y `str.endsWith(substr, [from])`: Verifican si `str` comienza o termina con `substr`.

```
let str = "Widget con id";
console.log(str.includes("id", 2)); // true
```

```
console.log(str.startsWith("id")); // false
console.log(str.endsWith("id")); // true
```

Extraer subcadenas

- `str.slice(ini, [fin])` → Extrae una parte del `string` entre `ini` y `fin` (excluyendo `fin`). Admite índices negativos.

```
let str = "patata";
console.log(str.slice(0, 5)); // 'patat'
console.log(str.slice(0, 1)); // 'p'
console.log(str.slice(2)); // 'tata'
console.log(str.slice(-4, -1)); // 'tat'
```

- `str.substring(ini, fin)` → Similar a `slice`, pero intercambia `ini` y `fin` si `ini > fin`.

```
let str = "patata";
console.log(str.substring(0, 5)); // 'patat'
console.log(str.substring(0, 1)); // 'p'
console.log(str.substring(2)); // 'tata'
console.log(str.substring(-4, -1)); // ''
```

Otros métodos útiles

- `str.codePointAt(pos)` → Devuelve el código UTF-16 del carácter en `pos`.
- `String.fromCodePoint(cod)` → Crea un carácter a partir de su código UTF-16.

```
let str = "patatA";
console.log(str.codePointAt(1)); // 97
console.log(str.codePointAt(5)); // 65
console.log(String.fromCodePoint(97)); //'a'
```

- `str.trim()` → Elimina los espacios en blanco al inicio y final del `string`.
- `str.repeat(n)` → Repite la cadena `n` veces.

```
let str = "  a  ";
console.log("*" + str + "*"); // '*  a  *'
console.log("*" +str.trim() + "*"); // '*a*'
console.log("dime".repeat(3));
// 'dimedimedime'
```

- `str.split(separador)` → Divide el `string` en un array usando el `separador`.

```
let str = "mi casa";

let subcadenas = str.split("a");
console.log(subcadenas); // ["mi c", "s", ""]
```

```
let palabras = str.split(" ");
console.log(palabras); // ["mi", "casa"]

let letras = str.split("");
console.log(letras); // ["m", "i", "c", "a", "s", "a"]
```

`Arrays`

Los arrays son objetos optimizados para almacenar datos indexados, permitiendo la manipulación eficiente. Al ser objetos:

- Se copian y asignan por referencia.
- No se comparan directamente con `==`.

Declaración de `arrays`

```
let arr = new Array();
let arr2 = [];
let arr3 = new Array();

let frutas = ["Manzanas", "Peras", "Ciruelas"];
let arr4 = new Array("Manzanas", "Peras", "Ciruelas");
```

Características principales

- Los arrays pueden almacenar elementos de cualquier tipo:

```
arr = ["Casa", 23, { nombre: "juan", edad: 45 }];
```

- Se permite la coma residual:

```
arr = ["Casa", 23, { nombre: "juan", edad: 45 },];
```

Métodos básicos

- `toString()` → convierte los elementos en una cadena separada por comas.

```
console.log(frutas.toString());
```

- Acceder y modificar elementos:

```
console.log("Elemento 0: " + frutas[0]);
frutas[1] = "Melocotones";
```

- Agregar elementos:

```
frutas[3] = "Uvas";
console.log("Después de cambiar dos elementos: " + frutas);
```

- Longitud del array: `.length`

```
console.log("Longitud del array: " + frutas.length);
```

Métodos de manipulación

- `pop()` → Elimina y devuelve el último elemento.

```
let ultimaFruta = frutas.pop();
console.log(ultimaFruta); // "Uvas";
console.log(frutas); // Manzanas,Peras,Ciruelas
```

- `push(elemento)` → Añade un elemento al final del array.

```
frutas.push("Melón");
console.log(frutas); // Manzanas,Peras,Ciruelas,Melón
```

- `shift()` → Elimina y devuelve el primer elemento.

```
let primeraFruta = frutas.shift();
console.log(primeraFruta); // "Manzanas";
console.log(frutas); // Manzanas,Peras,Ciruelas,Melón
```

- `unshift(elemento)` → Añade un elemento al inicio.

```
primeraFruta = frutas.unshift("Sandía");
console.log(frutas); // Sandía,Manzanas,Peras,Ciruelas,Melón
```

Bucle `for` y `for of`

```
for (let index = 0; index < frutas.length; index++) {
  console.log(frutas[index]);
}

for (const fruta of frutas) {
  console.log(fruta);
}
```

Propiedad `.length`

- Es el índice más alto + 1. Se puede modificar para truncar o vaciar el array. Si se asigna el valor 0, se elimina el array.

```
frutas[100] = "plátano";
console.log(frutas.length); // 101
console.log(frutas[99]); // undefined
frutas.length = 0; // Eliminamos el array
```

Operaciones avanzadas y estructuras especiales en Arrays

- `delete array[ pos ]` → Eliminar un elemento del array (desaconsejado)

```
delete frutas[9]; // Deja el hueco: undefined
```

- `new Array( num )` → Crea un array de una longitud concreta

```
arr = new Array(3);
console.log(arr.length); // 3
console.log(arr[0]); // undefined
```

- Arrays multidimensionales

```
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9],
];
console.log("Matriz bidimensional: " + matrix);
console.log("Acceder al elemento [1][1]: " + matrix[1][1]); // 5, el eleme
```

Métodos de arrays

Agregar o eliminar items de un array

- `arr.splice()` → Devuelve un array con los elementos eliminados

```
subArr = arr.splice(ini [, numBorrar, ele1, ele2…])
```

  - Comienza en la posición `ini` (puede ser negativo).
  - Elimina `numBorrar` elementos.
  - Inserta los elementos `ele1`, `ele2`, ... en esa posición.
  - Devuelve un array con los elementos eliminados.

```
// Eliminar elementos
let arr = ["Yo", "estudio", "JavaScript"];
arr.splice(1, 1);
console.log(arr); // "Yo,JavaScript"

// Eliminar elementos e insertar otros
```

```
arr = ["Yo", "estudio", "JavaScript", "ahora", "mismo"];
let eliminado = arr.splice(0, 3, "a", "bailar");
console.log(arr); // "a,bailar,ahora,mismo"
console.log(eliminado); // "Yo,estudio,JavaScript"

// Sólo insertar elementos
arr = ["Yo", "estudio", "JavaScript"];
arr.splice(2, 0, "mucho");
console.log(arr); // "Yo,estudio,mucho,JavaScript"

arr = ["Yo", "estudio", "JavaScript"];
arr.splice(-1, 0, "mucho");
console.log(arr); // "Yo,estudio,mucho,JavaScript"
```

Obtener un subarray

- `arr.slice()` → Devuelve un subarray sin modificar el array original.

```
subArr = arr.slice([ini [, fin])
```

  - Extrae elementos desde la posición `ini` (por defecto 0, puede ser negativo).
  - Hasta la posición `fin` (por defecto el largo del array, puede ser negativo).
  - El array original no cambia.

```
arr = ["Yo", "estudio", "JavaScript"];
let sub = arr.slice(1, 2);
console.log(sub); //"estudio"

sub = arr.slice();
console.log(sub); //"Yo, estudio,JavaScript"
```

Concatenar arrays

- `arr.concat()` → Crea un nuevo array combinando el original con otros valores o arrays.

```
nuevoArr = arr.concat(arg1, arg2, ...)
```

```
let arr = [1, 2];

// crea un array a partir de: arr y [3,4]
console.log(arr.concat([3, 4])); // 1,2,3,4

// crea un array a partir de: arr y [3,4] y [5,6]
console.log(arr.concat([3, 4], [5, 6])); // 1,2,3,4,5,6

// crea un array a partir de: arr y [3,4], luego agrega los valores 5 y 6
```

```
console.log(arr.concat([3, 4], 5, 6)); // 1,2,3,4,5,6
console.log(arr); // 1,2
```

Obtener la posición de un item en un array

- `arr.indexOf(item, from)` → Devuelve la primera posición de `item`, o `-1` si no lo encuentra.
- `arr.lastIndexOf(item, from)` → Busca de atrás hacia adelante.

```
let fruits = ["Apple", "Orange", "Apple"];

console.log(fruits.indexOf("Apple")); // 0 (first Apple)
console.log(fruits.lastIndexOf("Apple")); // 2 (last Apple)
```

Verificar si un item está en un array

- `arr.includes(item, from)` → Devuelve `true` si el ítem está en el array.

```
let fruits = ["Apple", "Orange", "Apple"];
console.log(fruits.includes("Apple")); // true
```

Dar la vuelta a un array

- `arr.reverse()` → Invierte el orden de los elementos en el array original.

```
let arr = [1, 2, 3, 4, 5];
let nuevo = arr.reverse();

console.log(arr); // 5,4,3,2,1
console.log(nuevo); // 5,4,3,2,1

nuevo[0] = 100;
console.log(arr); // 100,4,3,2,1
console.log(nuevo); // 100,4,3,2,1
```

Convertir un array en una cadena

- `arr.join(separador)` → Devuelve una cadena con los elementos del array separados por `separador`.

```
let arr = ["Bilbo", "Gandalf", "Nazgul"];
let str = arr.join("-");
console.log(str); // Bilbo-Gandalf-Nazgul
```

Verificar si una expresión es un array

- `Array.isArray(exp)` → Devuelve `true` si la expresión es un array.

```
console.log(Array.isArray({})); // false
console.log(Array.isArray([])); // true
```

Recorrer un array

- `arr.forEach((item, index, array) => {...})` → Ejecuta una función para cada ítem.

```
let personajes = ["Bilbo", "Gandalf", "Nazgul"];
personajes.forEach((item, index, array) => {
  console.log(`${item} está en la posición ${index} de ${array}`);
});
```

Buscar el primer elemento que cumpla una condición

- `arr.find((item) => {...})` → Devuelve el primer ítem que cumple la condición o `undefined`.

```
let users = [
  { id: 1, name: "John" },
  { id: 2, name: "Pete" },
  { id: 3, name: "Mary" },
];

let user = users.find((item) => item.id > 2); // { id: 3, name: "Mary" }
console.log(user.name); // Mary
```

Buscar el índice del primer elemento que cumpla una condición

- `arr.findIndex((item) => {...})` → Devuelve el índice del primer ítem que cumple la condición o `-1`.

```
let users = [
  { id: 1, name: "John" },
  { id: 2, name: "Pete" },
  { id: 3, name: "Mary" },
];

let index = users.findIndex((item) => item.id > 2); // 2
console.log(index); // 2
```

- `arr.findLastIndex((item) => {...})` → Busca desde el final.

Filtrar elementos que cumplen una condición

- `arr.filter((item) => {...})` → Devuelve un subarray con los ítems que cumplen la condición.

```
let users = [
  { id: 1, name: "John" },
  { id: 2, name: "Pete" },
  { id: 3, name: "Mary" },
```

```
];

let subArray = users.filter(
      (item) => item.id >= 2);
//  [{ id: 2, name: "Pete" }, { id: 3, name: "Mary" }]
subArray.forEach((item) => console.log(item.name)); // Pete Mary
```

Mapear un array

- `arr.map((item) => {...})` → Devuelve un nuevo array con los resultados de aplicar la función a cada ítem.

```
let users = [
  { id: 1, name: "John" },
  { id: 2, name: "Pete" },
  { id: 3, name: "Mary" },
];

let iniciales = users.map((item) => item.name[0]); // ["J", "P", "M"]
console.log(iniciales); // J, P, M
```

Ordenar un array

- `arr.sort()` → Ordena el array original. Por defecto es alfanumérica, pero se puede pasar una función personalizada.

```
let users = [
  { id: 2, name: "Pete" },
  { id: 1, name: "John" },
  { id: 3, name: "Mary" },
];

let ordenar = users.sort((item1, item2) => {
  if (item1.name < item2.name) {
    return -1;
  } else if (item1.name > item2.name) {
    return 1;
  } else {
    return 0;
  }
});

users.forEach((item) => console.log(item.name)); // John Mary Pete
```

```
function compareNumeric(a, b) {
  if (a > b) return 1;
  if (a == b) return 0;
  if (a < b) return -1;
```

```
}

let arr = [1, 2, 15];
arr.sort();
console.log(arr); // 1, 15, 2

arr = [1, 2, 15];
arr.sort(compareNumeric);

console.log(arr); // 1, 2, 15
```

Reducir un array

`arr.reduce((acum, item) => {...}, valorIni)` → Acumula los resultados de aplicar la función a cada ítem, empezando por `valorIni`.