

OBJETOS  
PREDEFINIDOS



# UD2: Objetos predefinidos y estructuras definidas por el usuario

## Parte 1: Objetos predefinidos

María Rodríguez Fernández [mariarfer@educastur.org](mailto:mariarfer@educastur.org)



# A lo largo de este documento...

- Habrás recordado el concepto de **clase/objeto/POO**
- Conocerás las **propiedades y métodos** más interesantes de los siguientes objetos predefinidos de JavaScript:
  - String
  - Math, Number
  - Date
  - Array
  - Map, Set

# Objetos

- Un **objeto** encapsula un conjunto de datos relacionados entre sí de modo que los puedo tratar de manera conjunta
- Habitualmente en un objeto distinguimos:
  - Estado:
    - Contenido de las variables que lo forman
    - A dichas variables las llamamos **propiedades**
  - Comportamiento:
    - Acciones (funciones) que puedo realizar con él.
    - A las funciones asociadas a un objeto las llamamos **métodos**.

Clase: Agrupa un conjunto de objetos con estado y comportamiento común

# En general...

- Creación de un objeto (una instancia)
  - `var nombreObjeto=new NombreClase`
- Acceso a propiedades:
  - `nombreObjeto.propiedad`
- Acceso a un método de un objeto:
  - `nombreObjeto.método([parametros])`

# Creación de String

- Sintaxis tradicional

```
var miCadena="texto de la cadena";
```

- Creación alternativa (sintaxis de objeto)

```
var miCadena=new String("texto de la cadena");
```

Independientemente de la sintaxis usada podemos acceder a los métodos y propiedades

# Propiedades y métodos de **String**

- PROPIEDADES

<b>length</b>	Longitud de la cadena
---------------	-----------------------

- MÉTODOS

<b>charAt(pos)</b>	Devuelve el carácter ubicado en <b>pos</b>
<b>charCodeAt(pos)</b>	Devuelve el Unicode del carácter ubicado en <b>pos</b>
<b>fromCharCode(code)</b>	Convierte valores Unicode a caracteres
<b>indexOf(car)</b>	Devuelve la posición de la primera ocurrencia del carácter buscado por <b>car</b>
<b>lastIndexOf(car)</b>	Devuelve la posición de la última ocurrencia del carácter buscado por <b>car</b>

# Más métodos de `String`

<code>replace(c1,c2)</code>	Busca la subcadena <b>c1</b> y la reemplaza con <b>c2</b>
<code>search(c)</code>	Busca la subcadena <b>c</b> y devuelve la posición donde se encontró
<code>slice(inicio,fin)</code>	Extrae y devuelve la subcadena entre los índices <b>inicio</b> y <b>fin</b> .
<code>split(separador)</code>	Devuelve un array de subcadenas. El parámetro especifica el carácter a usar para la separación de la cadena.
<code>toLowerCase()</code>	Convierte la cadena a minúsculas
<code>toUpperCase()</code>	Convierte la cadena a mayúsculas

# String (Ejemplos)

```
var cadena="El parapente es un deporte de riesgo";  
console.log("La longitud de la cadena es "+cadena.length);  
console.log(cadena.toLowerCase());  
console.log(cadena.charAt(3));  
console.log(cadena.indexOf("pente"));  
console.log(cadena.slice(3,16));
```



La longitud de la cadena es 36

el parapente es un deporte de riesgo

p

7

parapente es





# EJERCICIO PROPUESTO



- Crea un programa que pida al usuario su nombre y apellidos y muestre:
  - El tamaño del nombre más los apellidos (sin contar espacios).
  - La cadena en minúsculas y en mayúsculas.
  - Que divida el nombre y los apellidos y los muestre en 3 líneas, donde ponga  
Nombre:  
Apellido 1:  
Apellido 2:
  - Una propuesta de nombre de usuario, compuesto por el nombre, la inicial del primer apellido y la inicial del segundo : *ej. Para María Rodríguez Fernández sería mariarf.*
  - Una propuesta de nombre de usuario compuesto por las dos primeras letras del nombre y de los dos apellidos: *ej. marofe.*
- **RETO EXTRA:** ¿Cómo se podría hacer para generar una contraseña que fuese el nombre, pero separando cada letra con el número que indica su posición?
  - *Ejemplo:* `console.log(creaPassword("Maria")); // imprime "M0a1r2i3a4"`

# Objeto Math

- Permite realizar operaciones matemáticas
- Es una clase **estática**
  - No tiene constructor: no creamos instancias de objetos de tipo Math

```
var x = Math.PI; // Devuelve el número PI
var y = Math.sqrt(16); // Calcula la raíz cuadrada de 16
```

# Propiedades/métodos de Math más usados

- PROPIEDADES

<b>PI</b>	Número PI (aprox. 3.14)
<b>SQRT2</b>	Raíz cuadrada de 2 (aprox. 1.41)

- MÉTODOS

<b>ceil(x)</b>	Número <b>x</b> redondeado al alza al siguiente entero
<b>floor(x)</b>	Número <b>x</b> redondeado a la baja al anterior entero
<b>round(x)</b>	Redondea <b>x</b> al entero más próximo
<b>random()</b>	Devuelve un numero aleatorio entre 0 y 1
<b>pow(x,y)</b>	Devuelve el resultado de <b>x</b> elevado a <b>y</b>
<b>sqrt(x)</b>	Raíz cuadrada de <b>x</b>
<b>max(x,y,z...n)</b> <b>min(x,y,z...n)</b>	Máximo/mínimo de los números que se pasan como parámetros

# Objeto Number

- Es un envoltorio numérico para tipos numéricos primitivos
- Propiedades estáticas:

<b>MAX_VALUE</b>	Número más alto posible
<b>MIN_VALUE</b>	Número más bajo posible
<b>NEGATIVE_INFINITY</b>	Infinito negativo (en caso de overflow)
<b>POSITIVE_INFINITY</b>	Infinito positivo (en caso de overflow)

- Algún método útil:

<b>toFixed(n)</b>	Devuelve el número usando <b>n</b> dígitos decimales. Si no se especifica <b>n</b> por defecto se devuelve el número entero.
<b>toString(B)</b>	Representación como cadena en base B

# Conversión entre String y Number



- De String a Number:

- FORMA COMÚN

```
let n = Number(s);
```

- FORMA “PRO”

```
let n = +s;
```

- De Number a String:

- FORMA COMÚN

```
let s = num.toString();
```

- FORMA “PRO”

```
let s = num + "";
```

# EJERCICIO PROPUESTO



- Crea una función que reciba un año, y devuelva el siglo al que pertenece.
- Prueba la función generando 100 números aleatorios entre 0 y 2023.
- **RETO EXTRA:** “Traduce” la función al idioma inglés, teniendo en cuenta que el ordinal en ese idioma tiene distintas terminaciones (st, nd, rd, th)
  - Ej. , 12th century, 19th century, 21st century



# Clase Date

- Se usa para trabajar con fechas y horas
- Constructores:

<code>Date()</code>	Crea un objeto <b>Date</b> con la fecha actual
<code>Date(cadena)</code>	Crea un objeto <b>Date</b> a partir de la información de cadena
<code>Date(a,m,d,h,m,s,ms)</code>	Crea un objeto <b>Date</b> a partir del año, mes, día, hora, minuto, segundo y milisegundo.
<code>Date(a,m,d)</code>	Crea un objeto <b>Date</b> a partir del año, mes y día.

# Métodos de Date

<b>setHours(), getHours()</b>	Cambia/devuelve la hora (0-23)
<b>setMilliseconds()/getMilliseconds()</b>	Cambia/devuelve los milisegundos (0-9999)
<b>setMinutes()/getMinutes()</b>	Cambia/devuelve los segundos (0-59)
<b>setMonth(), getMonth()</b>	Cambia/devuelve el mes (0-11)
<b>setSeconds()/getSeconds()</b>	Cambia/devuelve los segundos (0-59)
<b>setDate()/getDate()</b>	Cambia/devuelve el día del mes (1-31)
<b>getDay()</b>	Devuelve el día de la semana (0-6)
<b>getFullYear()</b>	Devuelve el año (4 dígitos)
<b>getTime()</b>	Devuelve los milisegundos desde el 1/01/1970
<b>toDateString(), toLocaleString(), toGMTString()...</b>	Métodos para mostrar la fecha como cadena usando diferentes formatos.



# EJERCICIO PROPUESTO

## Cumpleaños



- Crea un programa que muestre cuántos días quedan para tu cumpleaños y qué día de la semana será
  - Los datos de entrada serán el día y el mes (puedes pedirlos al usuario o simplemente definirlos en una constante)
- Pruébalo con fechas límite (hoy, mañana...)
- **RETO EXTRA**
  - Dar también las horas que quedan
  - Pedir fecha de nacimiento en lugar de cumpleaños



# Arrays o vectores

- Cuando definimos un array en JavaScript realmente estamos definiendo un objeto de la clase **Array**
- Podemos inicializarlo de distintas maneras:

```
var mascotas=[];
```

```
var mascotas=new Array();
```

```
var mascotas=["Pepi","Luci","Bom"];
```

```
var mascotas=new Array("Pepi","Luci","Bom");
```

```
var mascotas=new Array(10); /* Se inicializan los elementos a null*/
```

- En JS podemos tener **distintos tipos de datos** almacenados en cada posición del array

```
var mezcla=new Array("Pepi",2,true);
```

# Extraer partes de un array

- Es posible extraer partes de un array para guardar cada uno de ellos en una variable dependiendo de su posición

```
const nombres = [ "Manolito", "Menganito", "Antoñita" ];  
  
const [primerNombre, segundoNombre] = nombres  
console.log(primerNombre) //Devuelve 'Manolito'  
console.log(segundoNombre); //Devuelve 'Menganito'
```

# Recorriendo un Array

- Bucle for “tradicional”

- Recorre índice

```
for(i=0;i<diasLaborables.length;i++)  
    console.log(diasLaborables[i]);
```

- Bucle for..in

```
var diasLaborables=["L", "M", "X", "J", "V"];  
for(var indice in diasLaborables)  
    console.log("Indice "+indice+" Valor:"+diasLaborables[indice]);
```

- También es válido para recorrer objetos literales (arrays asociativos)

```
var traducciones={"L":"Mon","M":"Tue","X":"Wed","J":"Thu","V":"Fri"};  
for(var clave in traducciones)  
    console.log("Clave "+clave+" Valor:"+traducciones[clave]);
```

- Bucle for ..of

- Recorre contenido

```
for(var dia of diasLaborables)  
    console.log(dia);
```

# EJERCICIO PROPUESTO



- Hacer una función que reciba un array y retorne otro array con la misma cantidad de elementos, pero que cada elemento sea el tipo de dato del array original. Ejemplo:

```
console.log(transformarATipos([1,"casa", {}])); // imprime ["number",  
  "string", "object"]  
console.log(transformarATipos([function() {}, true])); // imprime  
  ["function", "boolean"]
```

# Objeto **Array**: Propiedades y métodos

- PROPIEDADES

<b>length</b>	Longitud del array
---------------	--------------------

- MÉTODOS

<b>concat(array2)</b>	Concatena con <b>array2</b> y devuelve una copia de los arrays unidos.
<b>join(separador)</b>	Une todos los elementos del array separados por <b>separador</b>
<b>reverse()</b>	Invierte el orden de los elementos del array
<b>toString()</b>	Convierte el array a cadena y devuelve el resultado
<b>sort()</b>	Ordena los elementos de un array

# Objeto **Array**: Más métodos

<b>slice([inicio[,fin]])</b>	Devuelve una copia de una parte del array empezando por <b>inicio</b> y acabando en <b>fin</b>
<b>splice(i,n,e1,e2...)</b>	Cambia el contenido de un array eliminando o añadiendo contenido. <b>i</b> indica a partir de donde se modifica el contenido. <b>n</b> indica el número de elementos a eliminar. En caso de ser 0, <b>e1, e2...</b> indica los elementos a añadir.
<b>pop()</b>	Elimina el último elemento del array y devuelve dicho elemento
<b>push(elemento)</b>	Añade elementos al final del array y devuelve el nuevo tamaño
<b>shift()</b>	Elimina el primer elemento del array y lo devuelve
<b>unshift(elemento)</b>	Añade un elemento al comienzo del array, devolviendo el nuevo tamaño





# Borrado de elementos en un Array

- Podemos borrar un elemento **sin reducir la longitud del array**:

- Asignándole el valor null o cadena vacía.
- Mediante el operador `delete`

- Uso: `delete array[i]`

```
delete diasLaborables[0];
```

```
> diasLaborables
```

```
↳ [undefined × 1, "Martes", "Miercoles", "Jueves", "Viernes"]
```

- Podemos eliminar un elemento o una secuencia de elementos ajustando el número de elementos usando el método `splice(pos, cantidad)`:

```
diasLaborables.splice(0,2);
```

```
diasLaborables
```

```
["Miercoles", "Jueves", "Viernes"]
```

# Función map

- Es una forma de recorrer Arrays, devolviendo además un valor

```
const nombres = [ "Manolito", "Menganito", "Antoñita"];

const nombresSaludados = nombres.map (function (nombre){
    return `Hola ${nombre}`;
})
console.log(nombres) //Devuelve ['Manolito', 'Menganito', 'Antoñita']
console.log(nombresSaludados); //Devuelve ['Hola Manolito', ...]
//No se modifica nombres, se crea un nuevo array nombresSaludados
```

- Lo mismo con función flecha (más habitual):

```
const nombresSaludados = nombres.map ((nombre)=>`Hola ${nombre}`)
```

# Función find

- Busca un elemento dentro de un array y retorna el primero que cumpla con la condición especificada en la **función callback**

```
const nombres = [ "Manolito", "Menganito", "Antoñita"];  
  
const nombreEncontrado = nombres.find(nombre=>nombre === 'Menganito')  
  
console.log(nombreEncontrado) //Devuelve Menganito
```

# Función filter

- Permite **filtrar** uno o más elementos de una colección más grande de elementos basándose en alguna condición/preferencia

```
const nombres = [ "Manolito", "Menganito", "Antoñita"];

const nombresFiltrados = nombres.filter(function(nombre){
  if (nombre!=="Menganito"){
    return nombre;
  }
})

//Equivalente a:
const nombresFiltrados = nombres.filter((nombre)=>nombre!=="Menganito")

console.log(nombresFiltrados)//Devuelve ['Manolito', 'Antoñita']
```

# Spread operator

- Conocemos un método para concatenar arrays:

```
const nombres = [ "Manolito", "Menganito", "Antoñita"];
const masNombres = ["Leopoldo", "Fulgencio"];

console.log(nombres.concat(masNombres));
//Devuelve ['Manolito', 'Menganito', 'Antoñita', 'Leopoldo',
'Fulgencio']
```

Los arrays **nombres** y **masNombres** no se alteran

- El operador **spread** puede hacer lo mismo pero con una sintaxis más intuitiva:

```
console.log([...nombres, ...masNombres]);
//Devuelve ['Manolito', 'Menganito', 'Antoñita', 'Leopoldo',
'Fulgencio']
```

# Spread operator con objetos

```
const usuario = {
  nombre: "Manolito",
  edad: 30
}

const nacimiento = {
  pais: "España",
  ciudad: "Oviedo"
}

const usuarioInfo = {
  ...usuario,
  ...nacimiento
}

console.log(usuario); //{nombre: 'Manolito', edad: 30}
console.log(nacimiento); //{pais: 'España', ciudad: 'Oviedo'}
console.log(usuarioInfo); //{nombre: 'Manolito', edad: 30, pais:
'España', ciudad: 'Oviedo'}
```

# EJERCICIO PROPUESTO Biblioteca (1 de 4)

- Crea cuatro objetos para guardar información de libros (almacena cada uno en una constante) con los siguientes campos:
  - Nombre: cadena
  - Color: cadena
  - Autor: cadena
  - Nº páginas: entero
  - Editorial: cadena
  - Forrado: booleano
  - Url cover: cadena url
  - Comprar: función que mostrará por consola el mensaje: “**Libro de aventuras del autor XXX comprado**” usando string literal ``${}`
  - `toString`: función que muestra nombre, autor, páginas y si está forrado:



```
Lagartijas-A.J.Perez(110)**false
```

# EJERCICIO PROPUESTO Biblioteca (2 de 4)

- Usa tres de las constantes para crear un **array** que represente una biblioteca y muéstrala con `console.log` (automáticamente usará el método `toString` que hemos definido).
- Recorre los libros de la biblioteca con `for...of` y muestra las editoriales
- Crea las siguientes funciones:
  - **CheckPages**: Función **flecha** que devuelve `true` si el número de páginas del libro que se le pasa es mayor que 150.
    - Usa en este caso la **desestructuración de objetos**, de forma que le pasamos el libro entero pero la función recibe las páginas
  - **CheckLibro**: Función **flecha** a la que se le pasa la biblioteca y un libro y devuelve `true` si la biblioteca contiene ese título.
    - Pruébalo con un libro de la biblioteca y con el que has dejado fuera
- Usa el *spread operator* `...` para añadir a la biblioteca el libro que teníamos sin agregar junto a los tres que tenía



# EJERCICIO PROPUESTO Biblioteca (3 de 4)

- Usa la función **checkPaginas** en el método **filter**, para obtener un nuevo array **LibrosLargos** que contenga libros con páginas>150
- Añade las siguientes funciones al ejercicio:
  - **checkAutor**: Recibe un autor y la biblioteca y usa el método **find** para devolver un libro de dicho autor.
    - Usa el operador ternario para comprobar si en la biblioteca hay o no libros de Tolkien
  - **ForraLibro**: Función flecha que, recibiendo la biblioteca, pone a true el campo forrado de todos los libros que contiene.
  - **PrestarLibro**: Recibe la biblioteca y el título del libro, y lo borra del array en caso de encontrarlo. Busca información sobre el método **findIndex**
  - **DevolverLibro**: Recibe la biblioteca y el libro y mete el libro en el array de biblioteca.

# Arrays multidimensionales

- Los arrays bidimensionales no existen de manera nativa en JS
  - Podemos crear un array que en sus posiciones contengan otros arrays.
  - Podemos entender los arrays bidimensionales como *arrays de arrays*.
  - Acceso: `nombre[indice1][indice2]`

# Arrays multidimensionales: Ejemplo

```
var diasLaborables=new Array();

diasLaborables[0]=new Array("Lunes", "Martes", "Miercoles", "Jueves", "Viernes");
diasLaborables[1]=new Array("Monday", "Tuesday", "Wednesday", "Thursday", "Friday");
diasLaborables[2]=new Array("Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi");
diasLaborables[3]=new Array("Montag", "Dienstag", "Mittwoch", "Donnerstag", "Freitag");

console.log("La semana empieza en "+diasLaborables[0][0]);
console.log("Week ends on "+diasLaborables[1][4]);
```

Lunes	Martes	Miercoles	Jueves	Viernes
Monday	Tuesday	Wednesday	Thursday	Friday
Lundi	Mardi	Mercredi	Jeudi	Vendredi
Montag	Dienstag	Mittwoch	Donnerstag	Freitag

# Arrays multidimensionales: Ver el contenido

- Recorrido con bucles:

```
for(i=0;i<diasLaborables.length;i++)  
  for(j=0;j<diasLaborables[i].length;j++)  
    console.log(" "+diasLaborables[i][j]);
```

- Depuración por consola: `console.table(array)`

```
> console.table(diasLaborables)
```

VM431:1

(index)	0	1	2	3	4
0	'Lunes'	'Martes'	'Miercoles'	'Jueves'	'Viernes'
1	'Monday'	'Tuesday'	'Wednesday'	'Thursday'	'Friday'
2	'Lundi'	'Mardi'	'Mercredi'	'Jeudi'	'Vendredi'
3	'Montag'	'Dienstag'	'Mittwoch'	'Donnerstag'	'Freitag'

```
▶ Array(4)
```

# EJERCICIO PROPUESTO

## Arrays bidimensionales



- Queremos almacenar usando una **tabla** los resultados obtenidos en las elecciones en Villaconejos, teniendo en cuenta:
  - Ha habido 5 sedes para votar (Ayuntamiento, Polideportivo, Instituto, Mercado y Colegio)
  - Se han presentado 4 partidos (Puede que Villaconejos (PV), Obreros de Villaconejos (OV), Villaconejos Por el Si (VpSI), Unión Progreso y Villaconejos (UPV)).
- Se solicitarán al usuario los votos por sede y partido.
  - Para simularlo, en su lugar se pueden generar aleatoriamente los votos correspondientes a cada partido (entre 5 y 10 votos).
- A continuación, se mostrarán por consola:
  - Una tabla con todos los colegios electorales y partidos, así como sus votos asociados.

# RETO EXTRA



> console.table(resultados)

VM252:1

(index)	0	1	2	3	4	5
0	' '	'Ayuntamiento'	'Polideportivo'	'Instituto'	'Mercado'	'Colegio'
1	'PV'	8	10	8	10	7
2	'OV'	10	8	7	9	6
3	'VpSI'	5	7	6	9	9
4	'UPV'	9	7	5	9	10

► Array(5)

- **RETO:** Calcular el número total de votos por partido y por sede
- **SUPER-RETO:** Mostrar el recuento de votos por partido de forma ordenada...
  - Lee la información de los apuntes EXTRA sobre funciones de ordenación
  - Puedes usar un array auxiliar para almacenar para cada partido, su número de votos



# Map y Set

- **Map** es un diccionario clave-valor donde **cualquier tipo** puede ser usado como clave
  - Es la mayor diferencia con los arrays asociativos, donde las claves solo pueden ser cadenas de texto
  - Con cualquier tipo nos referimos no sólo a cadenas, números... sino incluso objetos o funciones
- **Set** permite almacenar valores **únicos** de cualquier tipo, es decir no pueden estar duplicados



# Objeto **Map**: Propiedades y métodos

- PROPIEDADES
- MÉTODOS

<b>Size</b>	Número de valores en el mapa
-------------	------------------------------

<b>Map([conjunto])</b>	Constructor. Acepta un conjunto de pares-valor
<b>set(key, value)</b>	Añade nueva pareja clave-valor
<b>get(key)</b>	Obtiene el valor asociado a una clave
<b>delete(key)</b>	Borra una pareja clave-valor mediante la clave
<b>has(key)</b>	Comprueba si hay determinada clave en el mapa
<b>values()</b>	Devuelve los valores del mapa
<b>keys()</b>	Devuelve las claves del mapa
<b>entries()</b>	Devuelve un conjunto de matrices [key,value]
<b>clear()</b>	Elimina todos los valores del mapa



# Objeto **Map**: Ejemplo

```
let mapa = new Map();
mapa.set('1', 'str1'); // un string como clave
mapa.set(1, 'num1'); // un número como clave
mapa.set(true, 'bool1'); // un booleano como clave

// Map mantiene el tipo de dato en las claves, por lo
// que estas dos son diferentes:
alert( mapa.get(1) ); // 'num1'
alert( mapa.get('1') ); // 'str1'
alert( mapa.size ); // 3
```

- Recorrido:

```
for(var [clave, valor] of mapa) {
    console.log(clave + " = " + valor);
}
```

1	=	str1
1	=	num1
true	=	bool1

# EJERCICIO PROPUESTO



- Desarrolla el script que maneje un conjunto usuarios y contraseñas mediante un **Map**. La clave sería el nombre de usuario (el nickname) y el valor la contraseña. Desarrolla una aplicación que mediante un menú te permita:
  - Añadir usuarios (comprobando previamente si existe o no)
  - Eliminar un usuario
  - Mostrar la colección entera de usuarios y contraseñas

# Objeto **Set**: Propiedades y métodos

- PROPIEDADES

<b>size</b>	Número de valores en el mapa
-------------	------------------------------

- MÉTODOS

<b>add(element)</b>	Añade un nuevo valor
<b>delete(element)</b>	Borra un valor
<b>has(element)</b>	Comprueba si hay un elemento en el conjunto
<b>values()</b>	Devuelve un objeto iterable con cada uno de los valores del conjunto.
<b>clear()</b>	Elimina todos los valores del conjunto

# RETO EXTRA

## Evitar duplicados



- Dados los siguientes arrays:

```
let array = [100, 23, 23, 23, 23, 67, 45];  
let outputArray = [];
```

- Haz que en outputArray estén los mismos números, pero evitando duplicados. Al final del proceso, en este ejemplo, el array contendrá:

```
outputArray = [100, 23, 67, 45]
```

- Será necesario hacerlo de dos formas:
  - Una “manual” usando bucles para recorrer el array original, viendo si cada número ya lo tenemos en el de salida o no.
  - Otra buscando en internet información sobre el método from de Array y el objeto Set.

# ¿Qué tal ha ido?

