4. Colecciones de Datos

Colecciones de datos

Javascript: dos maneras de estructurar valores:

- Arrays: listas ordenadas
  - Almacenar varios elementos similares ordenadamente
  - Podemos meter y sacar elementos a voluntad
- Objetos: diccionarios de propiedades
  - Agrupar varios valores relacionados entre sí
  - Ordenarlos dándoles nombres propios
  - Propiedades y métodos

Arrays

Array = lista de elementos

```
[1, 2, 3, 4]
```

- Se representan con [y]
- Dentro, una lista de valores separados por comas
  - Los valores pueden ser de cualquier tipo!

Arrays

```
> var lista = [1, 2, 3, 4]
  undefined
> lista
  [1, 2, 3, 4]
```

```
> var comprar = ["huevos", "leche", "pan"]
  undefined
> comprar
  ["huevos", "leche", "pan"]
```

Arrays

En un array se puede meter cualquier valor de cualquier tipo

```
var cosas = [[1, 2, 3], function() { alert("hola"); }, undefined, false]
undefined
cosas
[▶ Array[3] , function () { alert("hola"); }, undefined, false]
```

Arrays

Los elementos “numerados” (empezando por 0)

```
["Mick Jagger", "Keith Richards", "Charlie Watts", "Ronnie Wood"]
```

0 1 2 3

Arrays

Para acceder al enésimo elemento de un array:

`miArray[n]`

- Siendo **n** el índice del elemento al que se quiere acceder
- El índice **n** tiene que ser un número!

```
var stones = ["Mick Jagger","Keith Richards", "Charlie Watts", "Ronnie Wood"]
undefined
stones[0]
"Mick Jagger"
stones[1]
"Keith Richards"
```

Arrays

Ejercicio: escribe un programa que

1. Cree un array con al menos 5 elementos
   - ➡ Mejor si es una cosa más original que los números del 1 al 5
2. Muestra, de uno en uno, los elementos del array por consola

   Una pista: utiliza un bucle `for` para recorrer el array

Arrays

Ejercicio: escribe un programa que

1. Cree un array con números del 1 al 5
2. Recorre el array, sustituyendo todos los números pares por la cadena: “aquí había un número par”

Arrays

Longitud del array: propiedad `length`

```
> var lista = []
  undefined
> lista.length
  0
> lista = ["a", "b", "c", "d"]
  ["a", "b", "c", "d"]
> lista.length
  4
```

Arrays

Ejercicio: escribe una función `copiarLista`

- Reciba un array como parámetro
- Devuelva un nuevo array con los mismos elementos

Arrays

Ejercicio: escribe una función `estaEnLaLista`

- Recibe dos parámetros:
  - el primero es un array
  - el segundo puede ser cualquier cosa
- Devuelve `true` si el elemento pasado como segundo parámetro aparece dentro del array
- Devuelve `false` en caso contrario

Arrays

Ejercicio: escribe una función `sonIguales` que...

- Recibe dos arrays como parámetros, `a` y `b`
- Si todos los elementos de a son iguales `a` los de `b`, devuelve `true`
  - una pista: `a == b` no funciona!
- En caso contrario devuelve `false`

```
> sonIguales([1,2,3], [1,2,3])
  true
> sonIguales(["a","b","c"], [45, 12])
  false
```

Arrays

Ejercicio: función `cuantasVeces`

- Recibe dos parámetros: un array y un elemento cualquiera
- Devuelve en número de veces que el elemento aparece en el array

```
> cuantasVeces([1, 2, 3, 2, 1, 2], 1)
2
> cuantasVeces([1, 2, 3, 2, 1, 2], 2)
3
> cuantasVeces([1, 2, 3, 2, 1, 2], 5)
0
```

Arrays

Añadir y sacar elementos de un array

- Podemos añadir o sacar elementos del principio o del final del array
- Del final:
  - push
  - pop
- Del principio:
  - shift
  - unshift

Arrays

Del final:

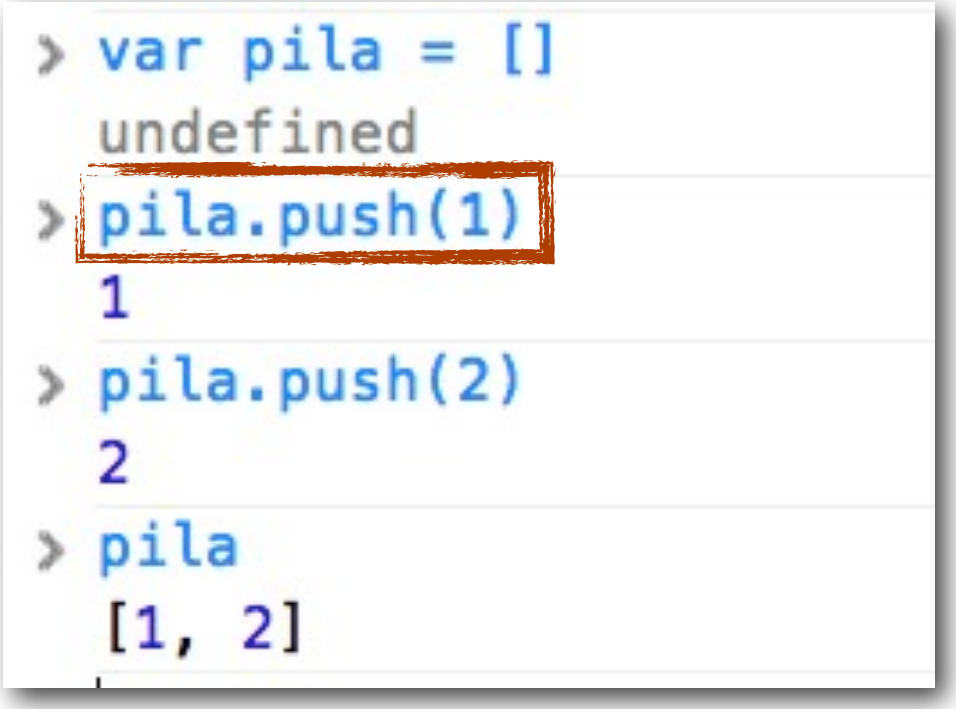
push: añadir un elemento al final del array (la más común)

```
> var pila = []
  undefined
> pila.push(1)
  1
> pila.push(2)
  2
> pila
  [1, 2]
```

Arrays

Del final:

push: añadir un elemento al final del array (la más común)



Arrays

Del final:

pop: sacar el último elemento del array

```
> pila
  [1, 2]
> pila.pop()
  2
> pila.pop()
  1
```

Arrays

Del principio:

unshift: añadir un elemento al principio

```
lista.unshift(1)
1
lista.unshift(2)
2
lista
[2, 1]
```

Arrays

Del principio:

shift: sacar el primer elemento del array

```
lista
[2, 1]
lista.shift()
2
lista.shift()
1
lista
[]
```

Arrays

Ejercicio: escribe un programa que...

- Cree un array vacío
- Meta 10 números aleatorios (0-100) en el array

Arrays

Ejercicio: función `eliminarElemento (lista, e)`

- Recibe un array como primer parámetro
- Recibe un elemento como segundo parámetro
- Devuelve un nuevo array los mismos elementos que le original pero sin el elemento `e`

```
> eliminarElemento([1, 2, 3, 4, 5], 5)
  [1, 2, 3, 4]
```

Arrays

Ejercicio: función `eliminaDuplicados`

Recibe un array como parámetro

- Devuelve un nuevo array los mismos elementos que le original pero sin ningún elemento duplicado
  - una pista: utiliza la función `estaEnLaLista`

```
eliminaDuplicados([1,1,1,1,1,1])
[1]
eliminaDuplicados([1,1,2, 3, 5, 2, 3, "jarl", 1,1,1,1])
[1, 2, 3, 5, "jarl"]
```

Arrays

Ejercicio (medio): mapear una función en un array

- Escribe una función **`mapear`** que:
  1. Reciba un array como primer parámetro
  2. Reciba una función como segundo parámetro
  3. Devuelva un nuevo array, construido a base de aplicar la función a cada uno de los elementos de array original

```
> function suma10 (n) { return n + 10; }
  undefined
> mapear([1, 2, 3, 4, 5], suma10)
  [11, 12, 13, 14, 15]
> mapear(["hola", "adios"], function(msg) { return msg + "!!"; })
  ["hola!!", "adios!!"]
```

Arrays

Ejercicio (difícil): ¿Qué hace este código?

```
function misterio (lista) {
  var nuevaLista = copiarLista(lista);
  var este;
  var siguiente;
  var terminado = false;
  while (!terminado) {
    terminado = true;
    for (var i=0; i< nuevaLista.length-1; i++) {
      este = nuevaLista[i];
      siguiente = nuevaLista[i+1];
      if (este > siguiente) {
        nuevaLista[i+1] = este;
        nuevaLista[i] = siguiente;
        terminado = false;
      }
    }
  }
  return nuevaLista;
}
```

Objetos

Objeto = colección de propiedades

- Se representan con { y }
  - ¡No confundir con bloques de código!
- Propiedad: "ranura" con nombre propio que puede contener cualquier valor

```
var objeto = {
  unaPropiedad: 1,
  otraPropiedad: "Robert Plant"
};
```

Objetos

Dos maneras de acceder a una propiedad:

- cadena entre `[` y `]`:

```
objeto["unaPropiedad"]
```

- punto:

```
objeto.unaPropiedad
```

Objetos

```
var objeto = { unaPropiedad: 1, otraPropiedad: "Robert Plant"}
undefined
objeto["unaPropiedad"]
1
objeto.otraPropiedad
"Robert Plant"
objeto
▶ Object
```

Objetos

Ejercicio: crea un objeto `usuario` que

Tenga 5 propiedades:

1. `nombre`: Un cadena de texto
2. `instrumentos`: Un array de cadenas
3. `diHola`: una función que muestre un `alert("hola")`
4. `followers`: un número
5. `activo`: un booleano

Objetos

```
var usuario = {
  nombre: "Ritchie Blackmore",
  instrumentos: ["Guitarra", "Bajo"],
  diHola: function() {
    alert("Hola!");
  },
  activo: false
};
```

Objetos

Un objeto puede contener otros objetos como propiedades

```
var rainbow = {
  guitarra: {
    nombre: "Ritchie Blackmore",
    desde: 1975,
    hasta: 1984
  },
  cantante: {
    nombre: "Ronnie James Dio",
    desde: 1975,
    hasta: 1979
  },
  bateria: {
    nombre: "Gary Driscoll",
    desde: 1975,
    hasta: 1987
  }
};
```

Objetos

Un objeto puede contener otros objetos como propiedades

Objetos

Ejercicio:

Crea un array con unos cuantos objetos que representen usuarios con la forma:

```
{
  nombre: "Mick",
  apellidos: "Jagger",
  edad: 69,
  activo: true
}
```

Crea una función `quitarInactivos` que cree un nuevo array que contenga solamente los usuarios activos

Objetos

Ejercicio:

Crea un array con unos cuantos objetos que representen usuarios con la forma:

```
{
  nombre: "Mick",
  apellidos: "Jagger",
  edad: 69,
  activo: true
}
```

Crea una función `ordenarPorNombre` que cree un nuevo array con los objetos ordenados alfabéticamente por nombre (pista: utiliza `eliminarElemento` y `copiarLista`)

Objetos

Ejercicio:

Crea un array con unos cuantos objetos que representen usuarios con la forma:

```
{
  nombre: "Mick",
  apellidos: "Jagger",
  edad: 69,
  activo: true
}
```

Crea una función `ordenarPor(lista, propiedad)` que ordene por `propiedad`

Objetos

```
> ordenarPor(rolling, "edad")
[▼Object , ▼Object , ▼Object , ▼Object ]
    activo: true          activo: true          activo: true          activo: false
    apellidos: "Wood"     apellidos: "Richards" apellidos: "Jagger"   apellidos: "Watts"
    edad: 65              edad: 68              edad: 69              edad: 71
    nombre: "Ronnie"      nombre: "Keith"       nombre: "Mick"        nombre: "Charlie"
  ▶__proto__: Object    ▶__proto__: Object    ▶__proto__: Object    ▶__proto__: Object
```

```
> ordenarPor(rolling, "apellidos")
[▼Object , ▼Object , ▼Object , ▼Object ]
    activo: true          activo: true          activo: false         activo: true
    apellidos: "Jagger"   apellidos: "Richards" apellidos: "Watts"    apellidos: "Wood"
    edad: 69              edad: 68              edad: 71              edad: 65
    nombre: "Mick"        nombre: "Keith"       nombre: "Charlie"     nombre: "Ronnie"
  ▶__proto__: Object    ▶__proto__: Object    ▶__proto__: Object    ▶__proto__: Object
```