Series favoritas

Supongamos que queremos representar al conjunto de nuestras series favoritas. ¿Cómo podríamos hacerlo?

```
let seriesFavoritasDeAna = ["Game of Thrones", "Breaking Bad", "House of Cards"];
let seriesFavoritasDeHector = ["En Terapia", "Recordando el Show de Alejandro Molina"]
```

Como ves, para representar a un conjunto de strings, colocamos todos esos strings que nos interesan, entre corchetes (`[` y `]`) separados por comas. Fácil, ¿no?

> Probá en la consola las siguientes consultas:
>
> - `seriesFavoritasDeAna`
> - `seriesFavoritasDeHector`
> - `["hola","mundo!"]`
> - `["hola","hola"]`

Consola Biblioteca

```
>
```

Commons Compartir-Igual, 4.0.

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas

Y esto, es una lista

Lo que acabamos de ver es cómo modelar fácilmente conjuntos de cosas. Mediante el uso de `[]`, en JavaScript contamos con una manera simple de agrupar esos elementos en listas.

¿Acaso hay una cantidad máxima de elementos? ¡No, no hay límite! Las listas pueden tener cualquier cantidad de elementos.

Y no sólo eso, sino que además, el orden es importante. Por ejemplo, no es lo mismo `["hola", "mundo"]` que `["mundo", "hola"]`: ambos tienen los mismos elementos, pero en posiciones diferentes.

Probá en la consola las siguientes consultas:

- `listasIguales(["hola", "mundo"], ["mundo", "hola"])`
- `listasIguales(["hola", "mundo"], ["hola", "mundo"])`
- `listasIguales(["hola", "mundo"], ["hola", "todo", "el", "mundo"])`
- `listasIguales(["hola"], ["hola", "mundo"])`
- `["hola", "mundo"] === ["mundo", "hola"]`
- `personas`
- `["mara", "julian"] === personas`
- `personas === personas`

¿Qué conclusiones podés sacar?

Consola Biblioteca

>

Juegos de azar

Pero, pero, ¿sólo podemos crear listas de strings? ¿Y si quiero, por ejemplo, representar los números de la lotería que salieron la semana pasada? ¿O las tiradas sucesivas de un dado? ¿O si salió cara o ceca en tiradas sucesivas de una moneda?

3. Juegos de azar

```
let numerosDeLoteria = [2, 11, 17, 32, 36, 39];
let tiradasDelDado = [1, 6, 6, 2, 2, 4];
let salioCara = [false, false, true, false];
```

Como ves, también podemos representar conjuntos de números o booleanos, de igual forma: escribiéndolos entre corchetes y separados por comas. Podemos tener listas de números, de strings, de booleanos, etc. ¡Incluso podríamos tener listas de listas!

Veamos si queda claro. Probá en la consola las siguientes consultas:

- `numerosDeLoteria`
- `salioCara`
- `[[1, 2, 3], [4, 5, 6]]`

Consola Biblioteca

>

Listas vacías

Genial, ¡parece que una lista puede contener cualquier tipo de elemento! Podemos tener listas de booleanos, de números, de strings, de listas...

4. Listas vacías

Y no sólo eso, sino que además pueden contener cualquier cantidad de elementos: uno, dos, quince, cientos.

¿Podremos entonces tener listas vacías, es decir, que no tengan elementos? ¡Por supuesto!

```
let unaListaVacia = []
```

> Probá escribir en la consola `unaListaVacia`

Consola Biblioteca

```
>
```

Términos y Condiciones

Reglas del Espacio de Consultas

¿Cuántos elementos tenés?

Por el momento ya sabemos qué cosas podemos representar con listas, y cómo hacerlo. Pero, ¿qué podemos hacer con ellas?

5. ¿Cuántos elementos tenés?

Empecemos por lo fácil: saber cuántos elementos hay en la lista. Esto lo podemos hacer utilizando la función `longitud`, de forma similar a lo que hacíamos con los strings.

Realizá las siguientes consultas en la consola:

- `longitud([])`
- `longitud(numerosDeLoteria)`
- `longitud([4, 3])`

Consola Biblioteca

>

Agregando sabor

Las listas son muy útiles para contener múltiples elementos. ¡Pero hay más! También podemos agregarle elementos en cualquier momento, utilizando la función `agregar`, que recibe dos parámetros: la lista y el elemento.

Por ejemplo:

```
let pertenencias = ["espada", "escudo", "antorcha"];
//longitud(pertenencias) devuelve 3;

agregar(pertenencias, "amuleto mágico");
//ahora longitud(pertenencias) devuelve 4
```

Como vemos, `agregar` suma un elemento a la lista, lo cual hace que su tamaño aumente. ¿Pero en qué parte de la lista lo agrega? ¿Al principio? ¿Al final? ¿En el medio?

> Averigualo vos: inspeccioná en la consola qué elementos contiene `pertenencias`, agregale una `"ballesta"` y volvé a inspeccionar `pertenencias`.
> Además existe un procedimiento `remover`, que recibe la lista y un elemento por parámetro. Investigá en la consola qué hace.

Consola | Biblioteca

```
>
```

Trasladar

Bueno, ya hablamos bastante; ¡es hora de la acción !

7. Trasladar

Definí el procedimiento `trasladar`, que tome dos listas y un elemento de la primera. `trasladar` debe sacar el elemento de la primera lista y agregarlo en la segunda.

Ejemplo:

```
let unaLista = [1, 2, 3];
let otraLista = [4, 5];

trasladar(unaLista, otraLista, 2);

unaLista //debería ser [1, 3]
otraLista //debería ser [4, 5, 2]
```

¡Dame una pista!

Solución | Biblioteca | Consola

```
1 function trasladar(list1, list2, elemento) {
2   agregar(list2, elemento);
3   remover(list1, elemento);
4 }
5
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡Felicitaciones!

Hasta ahora anduvimos agregando, quitando y consultando longitudes. ¿Qué más podemos hacer con las listas? ¡Seguinos!

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas

¿Y dónde está?

Otra cosa que queremos hacer con las listas es saber en qué posición se encuentra un elemento. Para ello utilizamos la función `posicion` de la siguiente manera:

```
posicion(["a", "la", "grande", "le", "puse", "cuca"], "grande"); //devuelve 2

let diasLaborales = ["lunes", "martes", "miercoles", "jueves", "viernes"]
posicion(diasLaborales, "lunes"); //devuelve 0
```

Como ves, lo curioso de esta función es que pareciera devolver siempre uno menos de lo esperado. Por ejemplo, la palabra `"grande"` aparece tercera, no segunda; y `"lunes"` es el primer día laboral, no el cero. ¿Es que los creadores de JavaScript se equivocaron?

¡No! Se trata de que en JavaScript, al igual que en muchos lenguajes, las posiciones de las listas arrancan en 0: el primer elemento está en la posición 0, el segundo en la 1, el tercero en la 2, y así.

¿Y qué sucede si le pasás como argumento a `posicion` un elemento que no tiene? ¡Averigualo!

Probá lo siguiente:

```
posicion(diasLaborales, "osvaldo")
```

Consola Biblioteca

```
>
```

Contiene

¡Ahora te toca a vos!

Definí la función `contiene` que nos diga si una lista contiene un cierto elemento.

```
> contiene([1, 6, 7, 6], 7)
true
> contiene([1, 6, 7, 6], 6)
true
> contiene([], 7)
false
> contiene([8, 5], 7)
false
```

¡Dame una pista!

Solución | Biblioteca | Consola

```
1 function contiene(unaLista,elemento){
2   return posicion(unaLista,elemento)>=0;
3 }
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡Bien hecho!

Si venís prestando atención a los ejemplos de consulta, habrás notado que las listas también pueden tener elementos duplicados: `[1, 2, 1]`, `["hola", "hola"]`, etc.

Por tanto, `posicion` en realidad devuelve la posición de la *primera aparición* del elemento en la lista. Por ejemplo:

```
> posicion(["qué", "es", "eso", "eso", "es", "queso"], "es")
1 //devuelve 1 porque si bien "es" también está en la posición 4, aparece primero en la posición 1.
```

Términos y Condiciones

Reglas del Espacio de Consultas

Enésimo elemento

Así como existe una función para averiguar en qué posición está un elemento, también puede ocurrir que queramos saber lo contrario: qué elemento está en una cierta posición.

Para averiguarlo podemos usar el **operador de indexación**, escribiendo después de la colección y entre corchetes `[]` la posición que queremos para averiguar:

```
> mesesDelAnio[0]
"enero"
> ["ese", "perro", "tiene", "la", "cola", "peluda"][1]
"perro"
```

¡Ojo! El número que le pases, formalmente llamado **índice**, debe ser menor a la longitud de la lista, o cosas malas pueden suceder.

> Probalo en la consola: ¿qué sucede si le pedís el elemento 0 a una lista vacía? ¿O si le pedís el elemento 48 a una lista de 2 elementos?

Consola | Biblioteca

```
>
```

Commons Compartir-Igual, 4.0.

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas

Más premios

Si le pedís un elemento en una posición igual o mayor al tamaño de la lista, vas a obtener `undefined`. No parece algo terrible, pero el problema es que con `undefined` no podés hacer nada realmente útil.

Así que la advertencia es: ¡no te pases de índice!

> Teniendo esto en cuenta, va un desafío: definí nuevamente la función `medallaSegunPuesto`, pero esta vez usando como máximo un único `if`. Quizás las listas te pueden ser útiles acá.
> Te recordamos qué hace la función: tiene que devolver la medalla que le corresponde a los primeros puestos de una competencia.
>
> ```
> > medallaSegunPuesto(1)
> "oro"
> > medallaSegunPuesto(2)
> "plata"
> > medallaSegunPuesto(3)
> "bronce"
> > medallaSegunPuesto(4)
> "nada"
> > medallaSegunPuesto(5)
> "nada"
> ```

¡Dame una pista!

Solución | Biblioteca | Consola

```
1 function medallaSegunPuesto(puesto){
2   let unaLista = ["oro","plata","bronce"];
3   if (puesto<=(longitud(unaLista)) && puesto>=1) {
4      return unaLista[puesto-1];
5   } else {return "nada"}
6 }
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

No te olvides de saludar

Vamos a conocer una manera de recorrer los elementos de una lista con un nuevo amigo: el `for`.

Imaginémonos que tenemos una lista con los precios de los productos que compramos en el supermercado y queremos restar cada uno de ellos a `plataEnBilletera`. Usando `for` podemos hacerlo así:

```
for(let precio of [10, 100, 87 ]) {
  plataEnBilletera  = plataEnBilletera - precio
}
```

donde `plataEnBilletera` es una variable que se va modificando a medida que recorremos los `precio`s.

Si teníamos $500 en nuestra billetera, después del `for` nos van a quedar $303 porque:

- Al principio `plataEnBilletera` era 500 y el primer `precio` de la lista es 10. Luego de hacer 500 - 10, `plataEnBilletera` es 490.
- A los 490 que quedaron en nuestra billetera, le restamos el segundo precio de la lista: 100. Ahora `plataEnBilletera` es 390.
- El último precio a restar es 87, por lo que, al hacer 390 - 87, la variable `plataEnBilletera` terminará siendo 303.

Completá el procedimiento `saludar` que recibe una lista de personas e imprime un saludo con `"hola"` para cada una de ellas.

```
> saludar(["Don Pepito", "Don Jose"])
hola Don Pepito
hola Don Jose

> saludar(["Elena", "Hector", "Tita"])
hola Elena
hola Hector
hola Tita
```

¡Dame una pista!

Solución | Biblioteca | Consola

```
1 function saludar(personas) {
2   for(let persona of personas) {
3     imprimir ("hola "+persona);
4   }
5 }
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas