Guía de ejercicio 4 - Introducción a JavaScript

 ¡Hola! Te damos la bienvenida a esta nueva guía de ejercicios.

¿En qué consiste esta guía?

En la siguiente guía podrás trabajar los siguientes aprendizajes:

- Acceder a valores almacenados en un arreglo
- Mostrar los elementos de un arreglo por consola utilizando un ciclo
- Agregar elementos a una lista HTML a partir de un arreglo de Strings
- Crear un objeto con propiedades a partir de la interpretación de un texto
- Crear el template de una tarjeta que contenga propiedades de un objeto
- Agregar los elementos de una lista HTML a partir de un arreglo de objetos

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Arreglos

Los arreglos son un tipo de dato que nos permite almacenar de manera ordenada un conjunto de elementos o datos. Este tipo de datos es útil cuando queremos guardar una lista de elementos, por ejemplo, un listado de nombres, una lista de compras, etc.

Para crear un arreglo se deben ocupar los corchetes [], y posteriormente declarar los elementos separados por comas.

```
const sucursales = ["Rebeca Mate 18", "Libertad 5", "Av manquehue Sur 31"]
const codigosDeDescuento = ["ABC","FE1","8IA"]
const clientesPremium = ["Andrea", "Fernando", "Mariel"]
```

Una buena práctica al momento de crear los arreglos es asignar un nombre en plural, ya que representan la colección de varios elementos, asignarles un nombre que represente varias entidades mejoran la interpretación en el código.

Para identificar en qué momento corresponde idealmente ocupar los arreglos se debe detectar una necesidad de coleccionar o agrupar información que esté relacionada entre sí.

Por ejemplo según la siguiente frase:

"Necesito agrupar todos los nombres de mis próximos pacientes:

En este caso podemos optar por crear un arreglo que almacena varios Strings y cuyo nombre represente su contenido:

```
const nombresDeMisPacientes = ["Fernanda", "José", "Mauricio", "Victor"]
```

Otro caso puede ser querer representar la siguiente lista:

Las comunas a las que podemos entregar comida a domicilio son:

- Providencia
- Santiago
- Estación Central
- Ñuñoa
- Independencia
- Recoleta

Entonces podemos crear el siguiente arreglo:

```
let comunasDisponiblesParaDelibery = [
      "Providencia",
      "Santiago",
      "Estación Central",
      "Ñuñoa",
      "Independencia",
      "Recoleta"
]
```

Acceder a los elementos de un arreglo

Para acceder a los elementos de un arreglo debemos escribir el nombre de la variable y acompañarla con los corchetes y dentro, el índice al que queremos acceder.

Por ejemplo, dado el siguiente arreglo:

```
const bancos = ["Estado","Santander","BCI", "ITAU", "Scotiabank"]
```

Para acceder al primer elemento, escribimos lo siguiente:

```
bancos[0] // "Estado"
```

Podrías estar preguntándote ¿Por qué para acceder al primer elemento se debe ocupar el índice cero? Sucede que los arreglos inician la cuenta de índices en cero.

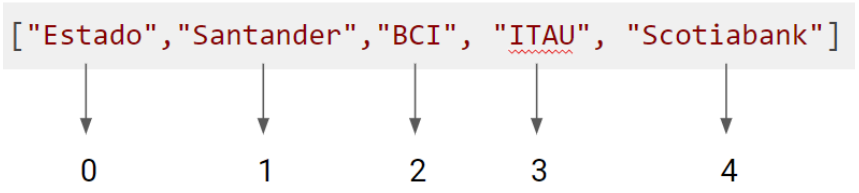

Imagen 1. índices del arreglo de bancos
Fuente: Desafío Latam

Para comprobarlo, en la consola del inspector de elementos crea el arreglo *bancos* y luego usa el `console.log()` para mostrar por consola el 2do elemento (índice 1):

```
console.log(bancos[1]) // "Santander"
```

¡Muy bien! Obtuvimos el banco “Santander”

Índices fuera de rango

¿Qué hubiera pasado si hubiésemos escrito un número de índice negativo o mayor a los que representan un elemento en el arreglo? La respuesta es que obtendremos un *undefined*, y esto lo puedes comprobar si intentas ejecutar cualquiera de las siguientes líneas:

```
console.log(bancos[6]) // undefined
console.log(bancos[-2]) // undefined
```

Cuando los índices son menores que cero o mayor que que la cantidad de elementos decimos que estos están fuera de rango.

No siempre sabremos de antemano cuántos elementos tiene el arreglo, puesto que un usuario a través de acciones podría agregar elementos o borrarlos.

Longitud del arreglo

Para conocer la cantidad de datos que tiene un arreglo arreglo.length

```
console.log(bancos.length) // 5
```

Con esta información podemos acceder al último elemento de un arreglo independiente del largo utilizando el índice arreglo.length - 1

```
const colores = ["Azúl", "Verde", "Morado", "Rojo", "Blanco",
"Amarillo"]
let longitudDelArregloColores = colores.length
console.log(colores[longitudDelArregloColores -1]) // "Amarillo"
```

Actividad 1: Accediendo a los nombres

Realicemos el siguiente ejercicio en donde a partir del siguiente arreglo:

```
let nombres = [
  "Juan",
  "Luisa",
  "Fabian",
  "Jorge",
  "Elon",
  "Steve",
  "Bill",
  "Estefany"
]
```

Utilizando el arreglo de la forma que aprendimos:

- Muestra por consola el nombres Juan utilizando el índice correspondiente
- Muestra por consola el nombres Steve utilizando el índice correspondiente
- Muestra el último elemento utilizando el índice consola el nombres Steve utilizando el índice correspondiente

Introduccion a ciclos

Los ciclos son otra forma de control de flujo que nos permiten repetir una acción hasta que se cumpla una condición:

Existen varias formas de hacer ciclos, partiremos con for of

En un caso hipotético, si tenemos un arreglo de alumnos y quisiéramos mostrar por consola el promedio de todos los estudiantes, escribiríamos algo como lo siguiente:

```
calcularPromedio(alumno[0]);
calcularPromedio(alumno[1]);
calcularPromedio(alumno[2]);
calcularPromedio(alumno[3]);
calcularPromedio(alumno[4]);
calcularPromedio(alumno[5]);
calcularPromedio(alumno[6]);
calcularPromedio(alumno[7]);
```

```
calcularPromedio(alumno[8]);
calcularPromedio(alumno[9]);
...
```

Suponiendo que no son 10 alumnos, sino 100 o 1000, esto ocuparía demasiadas líneas de código y tiempo de escritura innecesario. Para simplificar este proceso contamos con los ciclos, los cuales se pueden ocupar de diferentes formas.

Utilizando ciclos

Existen varias instrucciones que nos permiten hacer ciclos en Javascript. Partamos con for of.

For of

Con la instrucción *for of* podemos movernos a través de todos los elementos de un arreglo.

Veamos un ejemplo en donde dado el siguiente arreglo:

```
const estaciones = ["Verano", "Otoño", "Invierno", "Primavera"];
```

Ocupemos el for of para mostrar por consola todas las estaciones:

```
for(estacion of estaciones){
     console.log(estacion)
}
```

Cómo se puede apreciar no es necesario utilizar un índice. Cuando utilizamos for of estamos accediendo de forma secuencial al arreglo (pasito a pasito) mientras que cuando accedemos a través de un índice se dice que estamos accediendo de forma aleatoria, ya que podemos acceder a cualquier elemento en cualquier orden. Con for of accederemos ordenadamente y desde el primero al último de los elementos.

Palabras claves: Acceso secuencial. Acceso aleatorio

Creación de elementos HTML a partir de un arreglo con for of

Ahora que conocemos los arreglos, cómo crearlos y cómo acceder a sus elementos, veamos un caso donde agregaremos contenido en nuestra página web de forma dinámica a partir de un arreglo. Esto es muy importante porque en una próxima unidad aprenderemos cómo obtener este contenido de otros sitios webs.

Actividad 2: Creando elementos a partir de un arreglo

Crea el archivo nombres.html con la base del HTML, luego dentro del body agrega el siguiente script.

```
<ul id="nombres"> </ul>
<script>
  const data = ['Javiera', 'Camila', 'Francisco', 'Jorge', 'Daniela']
  const d = document.querySelector("#nombres")
  for (let item of data){
    d.innerHTML+= '<li>' + item + '</li>'
  }
</script>
```

Abre la página con el navegador y deberías observar los elementos agregados dinámicamente.

La línea `d.innerHTML+= 'valor'` se lee como concatena al innerHTML este valor nuevo después de los anteriores o el valor del innerHTML será equivalente al valor anterior más este nuevo contenido.

Actividad:

Crea una nueva página web llamada precios.html con la sección con id precios, y utilizando el siguiente arreglo `[1000, 2500, 3100, 4800, 7500]`. Agrega los elementos en distintos párrafos a la página web.

Evitando actualizar el DOM innecesariamente

Cada vez que le asignamos al .innerHTML un valor nuevo estamos forzando una actualización de la página, esto podemos evitarlo utilizando una nueva variable donde vayamos guardando la información y después del ciclo for actualizar la página, de esta forma solo la actualizaremos una vez

```
<ul id="nombres"> </ul>
<script>
  const data = ['Javiera', 'Camila', 'Francisco', 'Jorge', 'Daniela']
  const d = document.querySelector("#nombres")
  let html = ""
  for (let item of data){
```

```
      html += '<li>' + item + '</li>'
    }
    d.innerHTML = html
```

Es importante que la variable html la declaremos con let en lugar de `const`, puesto que el contenido de esta variable se irá modificando en cada iteración. Recordemos que las variables declaradas con const no pueden modificarse. De todas formas podemos probar cambiando la declaración a `const` y observar el error en la consola del inspector de elementos.

Actividad 3: Evitando modificar el DOM innecesariamente

Actualiza la actividad precios.html para evitar la actualización innecesaria del .innerHTML del elemento seleccionado.

Interpolación

Introducción a interpolación

La interpolación es otra forma de juntar texto con variables:

```
const nombre = "Luis"
const fecha = "2022-03-25"

const cabecera =
"Hola, mi nombre es "+ nombre +", hoy "+ fecha +" quiero emitir una solicitud"
const otra_forma =
`Hola, mi nombre es ${nombre}, hoy ${fecha} quiero emitir una solicitud`
```

Si hacemos un `console.log` de las variables en ambos casos obtendremos:

> *"Hola, mi nombre es Luis, hoy 2022-03-25 quiero emitir una solicitud"*

Podemos ocupar la interpolación para facilitar el armar plantillas o templates o sea textos similares con ligeras diferencias, como cuando enviamos un mail masivo pero cada uno de los emails tiene el nombre del cliente.

Sintaxis

Para interpolar se debe seguir la sintaxis, que consiste en usar las comillas invertidas (` `) y dentro de éstas podemos escribir texto y variables. Para utilizar variables debes envolverlas en símbolos de pesos y las llaves ${}. De esta manera no será necesario ocupar el símbolo (+) en nuestra composición, haciendo más cómoda la lectura y escritura de nuestro template.

Además dentro de las comillas invertidas podemos hacer saltos de línea sin ningún problema, lo cual nos ayuda a crear e interpretar los templates más leales a su forma final, como se puede ver en el siguiente código en donde se está creando un template con etiquetas HTML.

```
const template = `
    <h1>${titulo}</h1>
    <p>${parrafo}</p>
`
```

Con lo aprendido podemos mejorar nuestro programa en el archivo nombres.html utilizando interpolación

```
 <ul id="nombres"> </ul>
 <script>
   const data = ['Javiera', 'Camila', 'Francisco', 'Jorge', 'Daniela']
   const d = document.querySelector("#nombres")
   for (let item of data){
     d.innerHTML+= `<li> + ${item} + </li>` /* Aquí estamos interpolando
*/
   }
 </script>
```

Finalmente utilizar interpolación o no es una cosa de gustos, pero tiende a hacer el trabajo un poco más sencillo evitando tener que abrir y cerrar cada string mientras concatenamos variables. A partir de ahora ocuparemos la interpolación frecuentemente.

Ciclos y condiciones

En algunas ocasiones será conveniente utilizar condiciones dentro de los ciclos, digamos por ejemplo que tenemos un arreglo con distintas mediciones y queremos eliminar los valores extremos, supongamos que nuestros datos solo serán válidos si están entre 1000 y 2000, cualquier otro valor no lo mostraremos:

```
 <ul id="datos"> </ul>
 <script>
   const datos = [1200, 350, 1500, 1400, 250, 5000, 1950, 1952]
   const d = document.querySelector("#nombres")
   for (let valor of datos){
     if (valor > 1000 && valor < 2000){
       d.innerHTML+= `<li> + ${item} + </li>` /* Aquí estamos
interpolando */
     }
   }
 </script>
```

Actividad 4: Ciclos y condiciones

Crear el archivo misdatos.html donde utilizando el mismo arreglo del código anterior deberas mostrar de color verde todos los datos que cumplen la condición y de color rojo todos aquellos que no la cumplen. La condición es que los valores estén entre 1000 y 2000 para que se muestre en color verde.

Arreglos bidimensionales

Los arreglos pueden almacenar cualquier tipo de dato, incluso pueden almacenar otros arreglos.

```
const productos = [
  ["Patineta", "verde", 35990],
  ["Bicicleta", "Amarilla", 120990],
  ["Patines", "Morado", 60990],
  ["Scooter", "Negro", 250990]
];
```

Donde el elemento 0 es un arreglo que contiene `["Patineta", "verde", 35990]`. Si queremos mostrar el nombre del producto en la posición 1 “Bicicleta” debemos ejecutar la siguiente línea:

```
console.log(productos[1][0]); // "Bicicleta"
```

Modificando el DOM a partir de un arreglo bidimensional

Trabajar con arreglos bidimensionales es similar al arreglo unidimensional. Por ejemplo en el caso del arreglo de productos, tenemos que utilizar un ciclo para recorrer todos los productos, y dentro de ese ciclo accederemos a los elementos interiores.

```
const productos = [
  ["Patineta", "verde", 35990],
  ["Bicicleta", "Amarilla", 120990],
  ["Patines", "Morado", 60990],
  ["Scooter", "Negro", 250990]
];
let html = ''
for (let producto of productos) {
  html += `<div>
            <h1> ${producto[0]} </h1>
            <p> ${producto[1]} </p>
            <p> ${producto[2]} </p>`
}
const body = document.querySelector("body")
body.innerHTML = html
```

En algunos casos no sabremos cuántos elementos tiene un arreglo, por ejemplo supongamos que tenemos que mostrar los datos de un estudiante y notas y tenemos que mostrar todas las notas independiente de cuantas sean. Cómo aprendimos previamente eso se resuelve con ciclos, necesitamos un ciclo para recorrer el arreglo de estudiantes y otro para recorrer las notas.

```
const estudiantes = [
  ["Francisca", 10, 8, 10],
  ["Camila", 9, 9, 10, 9],
  ["Patricio", 7, 9, 9, 6, 10, 10],
  ["Pedro", 8, 8, 10]
];
let html = ""
for (let estudiante of estudiantes) {
  html += `<p>`
  for (let nota of estudiante){
    html += ` ${nota} `
  }
  html += `</p>`
}
const body = document.querySelector("body")
```

```
    body.innerHTML = html
```

Los arreglos bidimensionales nos permiten acumular información por grupos, sin embargo una desventaja importante es que los datos que se escriben y almacenan no tienen una representación literaria y depende estrictamente del orden en el que se asignan los datos para que exista una concordancia entre los subarreglos.

Para solventar este problema de identificación de datos y que toda información sea representada por un nombre o título, podemos ocupar otro tipo de datos conocidos como los objetos.

Objetos

Los objetos nos permiten agrupar la información por clave y valor.

Veamos un ejemplo en donde queremos guardar un auto marca Mazda, modelo 3 Sport y de origen Japonés.

Imagen 2. Representación de un objeto
Fuente: Desafío Latam

```
let automovil = {
  marca: "Mazda",
  modelo: "3 Sport",
  origen: "Japón",
}
```

De esta manera podemos agrupar toda la información de una misma entidad.

Una buena práctica al momento de crear objetos es asignar un nombre en singular, de preferencia un sustantivo que represente el conjunto de cualidades o funcionalidades que posea una misma entidad.

Para identificar en qué momento debemos ocupar un objeto podemos hacer un ejercicio en donde a partir de una frase se ubique el *sujeto* y toda la información que esté relacionada a éste, por ejemplo dada la siguiente frase:

> "Un teléfono de color blanco tiene android como sistema operativo y 6GB de ram"

Entonces podemos representar esta información con el siguiente objeto:

```
const telefono = {
  color: "blanco",
  so: "Android",
  ram: "6GB"
}
```

Actividad 5: Crea los objetos

Crea diferentes objetos que representen los siguientes textos:

- Una guitarra tiene 6 cuerdas, es de color azul y es del año 1994
- Una computadora ASUS tiene 16GB de RAM, tiene Windows y es de tipo gamer
- Una casa tiene 4 cuartos, 2 baños, 1 patio y 1 garage. Además está ubicada en la calle San Diego 8081

Accediendo a datos de un objeto

Existen 2 maneras de acceder a los datos de un objeto, las cuales son:

- Usando el punto (.)

```
automovil.marca
```

- Usando los corchetes []

```
automovil["marca"]
```

Ambas maneras pueden ser utilizadas, aunque esta segunda es más ocupada cuando necesitamos agregar dinámicamente el nombre de la propiedad.

Si quisiéramos mostrar la marca por consola podemos ocupar cualquiera de las siguientes líneas:

```
console.log(auto.marca);
```

```
console.log(auto["marca"]);
```

Actividad 6: Accediendo a los datos de un objeto

Según el siguiente objeto:

```
const persona = {
   nombre: "Pedro",
   apellido: "Perez",
   profesion: "Frontend Developer",
   hobby: "Trekking",
   añoDeNacimiento: 1988,
}
```

Muestra por consola el año de nacimiento y el apellido

Creación de templates con datos de un objetos

Ahora que entendemos que los objetos son la representación de una entidad, podemos crear templates (plantillas) que contengan los datos de un mismo objeto.

Por ejemplo, si tuviéramos que desarrollar un portal de noticias sobre la industria automotriz, los artículos pueden ser templates generados en base a la información guardada en un objeto.

Por ejemplo, teniendo una sección HTML que contenga los artículos a crear:

```
<section class="articulos"></section>
```

Podemos ocupar objetos como el siguiente:

```
const articulo = {
 id: 31,
 titulo: "Autos nuevos en Chile",
 subtitulo: "El mercado de autos se normaliza",
 descripcion: `No es novedad que los precios
 de los autos usados se han disparado debido
 a la falta en stock de autos nuevos, sin embargo
 puede que esto esté llegando a su fin...`
};
```

Para generar el template del artículo con los datos del objeto y posteriormente agregarlo a la sección:

```
const articulosSection = document.querySelector(".articulos")

articulosSection.innerHTML = `
    <article class="articulo">
        <h4>${articulo.titulo}</h4>
        <h6>${articulo.subtitulo}</h6>
        <p>${articulo.descripcion}</p>
        <a href="/articulo/${articulo.id}"><button>Ver más</button></a>
    </article>
`;
```

Y con un poco de estilos CSS:

```
.articulo {
 width: 250px;
 padding: 10px;
 background: lightgray;
}
```

Obtendremos un artículo con contenido dinámico y generado a partir de los datos de un objeto:

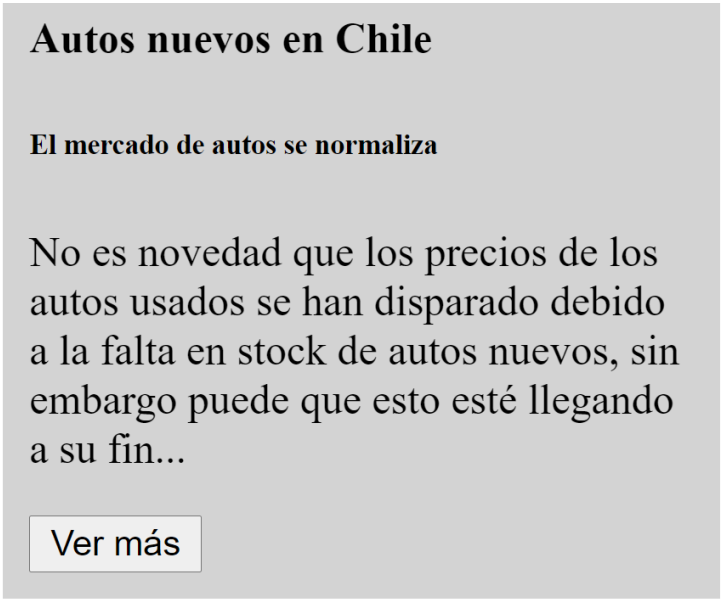

Imagen 3. Artículo generado con los datos de un objeto
Fuente: Desafío Latam

Actividad 5: Tarjeta de producto con los datos de un objeto

Crea la tarjeta de un producto en un ecommerce, a partir del siguiente objeto:

```
const producto = {
 id: 43,
 titulo: "Cafetera magnética",
 precio: 23990,
 color: "rojo",
 src: "...", // ingresa aquí la URL de la imagen,
 descripción: `Calienta tu café matutino
 con la nueva tecnología magnética`
};
```

1. Crea una sección en el HTML
2. Crea una variable que sea referencia de la sección HTML
3. Crea una variable que almacene el objeto del producto
4. Crea un template de una tarjeta usando la interpolación y los datos del producto
5. Utiliza el innerHTML para agregar la tarjeta a la sección creada

Arreglos de objetos

Ahora sabemos que podemos representar las características de una entidad en un objeto y que podemos acumular datos en un arreglo, lo siguiente será combinar ambos tipos de datos en un ejercicio donde necesitemos agrupar las ventas de la semana de una empresa.

Podemos representarlo con el siguiente arreglo de objetos:

```
const ventasDeLaSemana = [
  {dia: "Lunes", total: 34000},
  {dia: "Martes", total: 40000},
  {dia: "Miércoles", total: 41000},
  {dia: "Jueves", total: 50000},
  {dia: "Viernes", total: 62000},
  {dia: "Sábado", total: 85000},
  {dia: "Domingo", total: 20000},
]
```

Y a partir de este arreglo y la siguiente lista HTML:

```
<h3>Ventas de la semana:</h3>
<ul></ul>
```

Mostrarle al usuario de nuestras aplicaciones una lista interpolando los datos de cada venta, tal como se muestra a continuación:

```
const ul = document.querySelector("ul")

for(let venta of ventasDeLaSemana){
  ul.innerHTML+= `<li>${venta.dia}: ${venta.total} </li>`
}
```

Quedando como resultado lo siguiente:

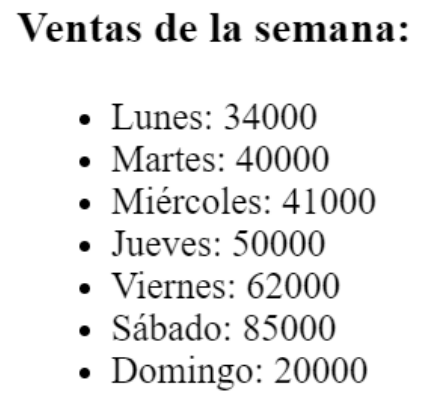

Imagen 4. Ventas de la semana
Fuente: Desafío Latam

Llenado de una tabla HTML

Sigamos practicando la utilidad de los arreglos de objetos ahora generando templates para llenar los registros de una tabla HTML que muestra los departamentos y los datos de contacto de un edificio de residencia, para esto ocupemos el siguiente código:

```
<table border="solid">
 <thead>
   <tr>
     <th>N°</th>
     <th>Propietario</th>
     <th>Número de contacto</th>
   </tr>
 </thead>
 <tbody></tbody>
</table>
```

Este llenado de registros consiste en agregar filas con la etiqueta tr al cuerpo de la tabla usando los datos del siguiente arreglo de objetos:

```
const departamentos = [
 { departamento: 323, propietario: "Natalia Jiménez", telft: "+56 9 5312 4578"},
 { departamento: 123, propietario: "Luis Fonsi", telft: "+56 9 4612 7894"},
 { departamento: 431, propietario: "David Bisbal", telft: "+56 9 8978 4465"},
 { departamento: 412, propietario: "Noel Schajris", telft: "+56 9 9874 6432"},
 { departamento: 203, propietario: "Ricardo Montaner", telft: "+56 9 8764 6813"},
]
```

Para esto, necesitaremos la referencia del cuerpo de la tabla (*tbody)* y finalmente, recorrer el arreglo de objetos de los departamentos agregando los templates de las filas

```
for (let dpto of departamentos) {
 const template = `
   <tr>
     <td>${dpto.departamento}</td>
     <td>${dpto.propietario}</td>
     <td>${dpto.telft}</td>
   </tr>
 `;
 tbody.innerHTML += template;
}
```

Y el resultado es el siguiente:

N°	Propietario	Número de contacto
323	Natalia Jiménez	+56 9 5312 4578
123	Luis Fonsi	+56 9 4612 7894
431	David Bisbal	+56 9 8978 4465
412	Noel Schajris	+56 9 9874 6432
203	Ricardo Montaner	+56 9 8764 6813

Imagen 9. Tabla de departamentos con un arreglo de objetos
Fuente: Desafío Latam

Actividad 8: Galería de iconos

Crea una galería de iconos utilizando el siguiente arreglo de objetos:

```
const iconos = [
 { icono: "🚗", descripcion: "Auto" },
 { icono: "🤖", descripcion: "Robot" },
 { icono: "👻", descripcion: "Fantasma" },
 { icono: "👽", descripcion: "Alien" },
 { icono: "🦷", descripcion: "Diente" },
 { icono: "🕹️", descripcion: "Joystick" }
];
```

1. Crea una sección HTML con la clase .iconos
2. Crea el arreglo de objetos con los iconos
3. Crea una variable que sea referencia de la sección de iconos
4. Recorre el arreglo de iconos y crea el template que desees para mostrar el ícono
5. Agrega en cada iteración el template creado en la sección de iconos del HTML

El resultado deberá ser el siguiente

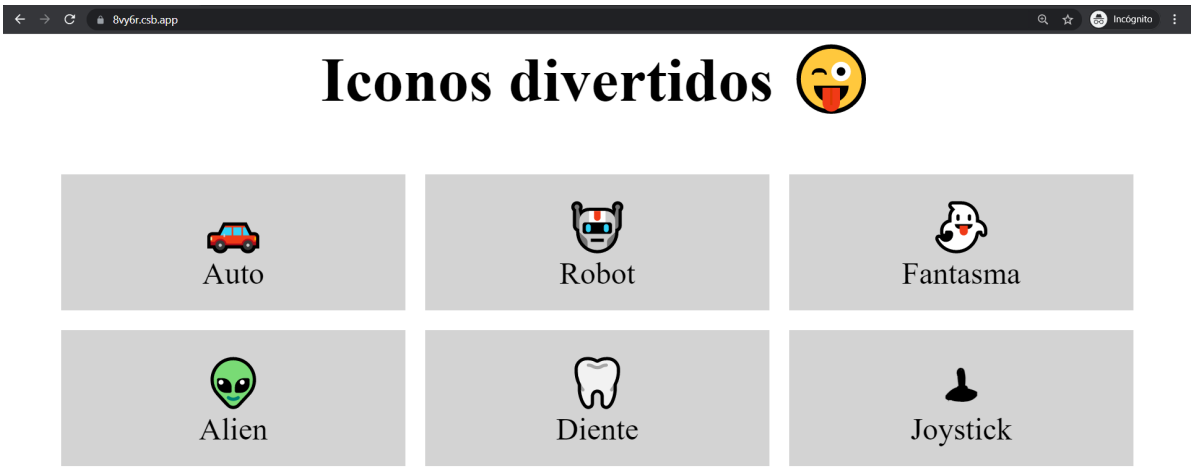

Imagen 5. Galería de iconos
Fuente: Desafío Latam

Resumen

- Los arreglos con colecciones de datos que nos ayudan a agrupar información
- Una buena práctica para asignar el nombre a los arreglos es ocupar siempre un sustantivo en plural
- Los arreglos pueden guardar cualquier tipo de datos y sus elementos son accesibles a partir de índices, iniciando en 0.
- Es posible acceder dinámicamente al último elemento de un arreglo si conocemos el largo del mismo, restándole a este valor 1 obtendremos el índice del último elemento
- Podemos ocupar los ciclos para recorrer los arreglos y acceder a sus diferentes elementos.

- Los objetos son un tipo de dato que almacenan todos los atributos de una entidad utilizando una estructura ***propiedad : valor***

- Los arreglos pueden almacenar otros arreglos y convertirse en un arreglo bidimensional, cuyos elementos se obtienen al unir 2 pares de corchetes, indicando un índice en cada uno

- Con los arreglos de objetos podemos agrupar entidades y con la ayuda de los ciclos, podemos iterarlos y generar templates para llenar una lista, tabla o incluso una galería de forma dinámica manipulando el DOM con *querySelector* e *innerHTML*.

Preguntas para preparar una entrevista laboral

- ¿Cuál es el problema de modificar .innerHTML dentro de un ciclo?

- ¿Si a = [1,2,3] qué se muestra con a[3]?

- ¿Cómo se obtiene el último elemento de un arreglo?