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

Un arreglo es una estructura de datos que nos brinda la capacidad de almacenar un conjunto de elementos de manera ordenada y acceder a ellos tanto de forma secuencial como aleatoria. Esta organización resulta útil cuando deseamos guardar y gestionar elementos como nombres, listas de compras, registro de notas, entre otros.

Para crear un arreglo se debe ocupar 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 arreglos es utilizar nombres en plural, ya que representan la colección de múltiples elementos. Esta convención contribuye a mejorar la comprensión del código.

Para identificar en qué momento corresponde ocupar 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 almacene varios Strings y cuyo nombre representa su contenido:

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

Otro caso puede ser querer representar un listado de comunas en las que las que podemos entregar comida a domicilio, tales como:

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

Entonces podemos crear el siguiente arreglo:

```
let comunasDisponiblesParaDelivery = [
```

```
      "Providencia",
      "Santiago",
      "Estación Central",
      "Ñuñoa",
      "Independencia",
      "Recoleta"
]
```

Acceder a los elementos de un arreglo

Para acceder a los elementos de un arreglo debemos utilizar el nombre de la variable seguido de corchetes, dentro de los cuales indicaremos 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? La razón es que los arreglos inician la cuenta de índices desde cero.

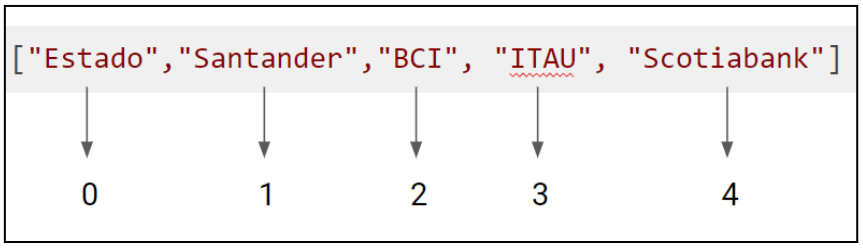

Imagen 1. índices del arreglo de bancos
Fuente: Desafío Latam

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

Nuestro arreglo de bancos contiene 5 elementos, por lo tanto, para acceder a sus elementos, debemos utilizar los índices del 0 al 4. ¿Qué hubiera pasado si hubiésemos escrito un índice mayor a 4 o un número negativo? La respuesta es que obtendremos un *undefined*, y esto lo puedes comprobar si intentas ejecutar cualquiera de las siguientes líneas en la consola:

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

Cuando los índices son menores que cero o mayor que la cantidad de elementos presentes en el arreglo 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 utilizaremos la propiedad **length**, que se aplica de la siguiente manera:

```
nombreArray.length

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

Con esta información podemos obtener el último índice de un arreglo independiente del largo, utilizando la operación arreglo.length - 1

```
const colores = ["Azúl", "Verde", "Morado", "Rojo", "Blanco",
"Amarillo"]
let longitudDelArreglo = colores.length // 6
let ultimoIndice = longitudDelArreglo - 1 // 5
console.log(colores[ultimoIndice]) // "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 nombre Juan utilizando el índice correspondiente.
- Muestra por consola el nombre Steve utilizando el índice correspondiente.
- Muestra el último elemento 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.
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 recorreremos el siguiente arreglo:

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

Ocuparemos la instrucción for of para mostrar por consola todas las estaciones:

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

Cuando utilizamos el ciclo "for of" estamos accediendo al arreglo de forma secuencial, mientras que al acceder a través de un índice se dice que estamos accediendo de forma aleatoria, ya que podemos acceder a cualquier elemento en cualquier orden. Con el ciclo "for of" accedemos a los elementos de forma ordenada, desde el primero hasta el último del arreglo.

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 desde 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+= '<li>' + item + '</li>'` se interpreta como: "concatena este nuevo valor al contenido actual de d.innerHTML, agregándolo después de los valores anteriores". En otras palabras, el valor del innerHTML será igual al valor previo más el nuevo contenido añadido.

Actividad 2.1: Crea una nueva página web llamada precios.html.

En el body crea una sección con id = precios, y utilizando el siguiente arreglo `[1000, 2500, 3100, 4800, 7500]` agrega los elementos en distintos párrafos.

Evitando actualizar el DOM innecesariamente

Cada vez que asignamos un valor nuevo al atributo innerHTML, estamos provocando una actualización de la página. No obstante, podemos evitar esta actualización utilizando una nueva variable para almacenar la información. Después de completar el ciclo, podemos realizar una única actualización de la página utilizando el contenido guardado en la variable. De esta manera, optimizamos el rendimiento de la página.

```
<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.

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

Hasta ahora hemos concatenado textos y variables de forma manual con el símbolo "+". La interpolación es otra forma de juntar texto con variables, pero de forma más concisa y legible, permitiendo que las variables se inserten directamente en la cadena de texto.

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

const concatenacionManual =
"Hola, mi nombre es "+ nombre +", hoy "+ fecha +" quiero emitir una solicitud"

const interpolacion =
`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"*

La interpolación resulta útil para simplificar la creación de plantillas o modelos de textos similares que requieren pequeñas variaciones, como cuando enviamos correos masivos personalizados para cada cliente. En estos casos, podemos utilizar la interpolación para insertar de manera sencilla el nombre del cliente u otras variables específicas en el contenido del correo.

Sintaxis

La interpolación se realiza siguiendo una sintaxis específica. Para ello, se utilizan comillas invertidas (` `) para delimitar el texto, dentro del cual se pueden incluir variables. Para insertar una variable, debes envolverla entre un símbolo de pesos y llaves de la siguiente manera: ${variable}
Esta forma de interpolación evita la necesidad de utilizar el símbolo (+) para concatenar las partes del texto, lo que hace la lectura de nuestra planilla más legible.

Dentro de las comillas invertidas (` `) también podemos realizar saltos de línea. Esto resulta especialmente útil al crear y leer plantillas que deben reflejar fielmente su estructura final. Por ejemplo, a continuación se muestra el siguiente código en donde se está creando un template con etiquetas HTML.

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

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 en
    vez de escribir '<li>' + item + '</li>'  */
  }
</script>
```

En última instancia, el uso de la interpolación es una cuestión de preferencia, pero tiende a simplificar el trabajo al evitar la necesidad de abrir y cerrar cada string mientras concatenamos variables. A partir de ahora ocuparemos la interpolación frecuentemente.

Ciclos y condiciones

En ciertas situaciones, resulta útil utilizar condiciones dentro de los ciclos. Por ejemplo, si tenemos un arreglo con diversas mediciones y deseamos eliminar los valores extremos, podemos establecer una condición. En este caso, supongamos que consideramos válidos únicamente los valores comprendidos entre 1000 y 2000 mientras que cualquier otro valor no será mostrado:

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

```
</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 tienen la capacidad de almacenar distintos tipos de datos, incluso otros arreglos.

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

En el ejemplo anterior, el elemento 0 corresponde a un arreglo que contiene `["Patineta", "verde", 35990]`. Si queremos acceder al primer nombre de este arreglo interno, debemos ejecutar la siguiente línea de código:

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

De esta manera, accedemos al primer elemento del arreglo externo y luego al primer elemento del arreglo interno.

Modificando el DOM a partir de un arreglo bidimensional

Cuando trabajamos con arreglos bidimensionales, el enfoque es similar al de los arreglos unidimensionales. En el caso del arreglo de productos antes visto, tenemos que utilizar un ciclo para recorrer todos los productos, y, dentro de ese ciclo, acceder a los elementos internos.

```
  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>
           </div>`;
}
const body = document.querySelector("body")
body.innerHTML = html
```

En ciertos casos, puede que no conozcamos la cantidad exacta de elementos en un arreglo. Por ejemplo, si necesitamos mostrar los datos de un estudiante junto con todas sus notas, independientemente de la cantidad de notas que tenga, podemos resolverlo utilizando ciclos. Cómo aprendimos previamente, 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 agrupar información de manera estructurada. Sin embargo, una desventaja importante es que los datos almacenados no tienen una representación explícita y dependen estrictamente del orden en el que se asignan los subarreglos. Esto puede dificultar la identificación y organización de los datos.

Para solventar este problema y lograr una mejor representación de la información mediante nombres o títulos, podemos utilizar otro tipo de estructura de datos conocida como objetos.

Objetos

Los objetos nos permiten agrupar la información utilizando pares de clave y valor.

Imaginemos que queremos almacenar información sobre un automóvil con las siguientes características: marca Mazda, modelo 3 Sport y de origen Japonés. Primero, crearemos un objeto llamado automóvil que contendrá 3 propiedades: “marca”, “modelo” y “origen”. Como mencionamos, los objetos están conformados por pares de clave y valor. En este caso, cada propiedad corresponde a una clave y el valor corresponde a la información específica para cada propiedad, es decir “Mazda”, “3 Sport” y “Japón”.

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. en un solo lugar.

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

Para determinar cuándo debemos utilizar un objeto, podemos realizar un ejercicio práctico en donde identificamos el *sujeto* de una frase y toda la información asociada a él. Por ejemplo, dada la siguiente frase:

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

En este caso, el sujeto de la frase es "teléfono" y sus propiedades son "color", "so" y "ram".

```
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 dos maneras de acceder a los datos de un objeto, las cuales son:

- Usando el operador punto (.)

```
automovil.marca //accedemos a la propiedad marca del objeto automovil
```

- Usando los corchetes [ ]

```
automovil["marca"] //para acceder a la propiedad, ingresamos su nombre
entre comillas al interior de los corchetes.
```

Ambas formas son válidas, sin embargo, la forma con corchetes es especialmente útil cuando necesitamos agregar dinámicamente el nombre de la propiedad, ya que nos brinda mayor flexibilidad al permitirnos utilizar expresiones o variables para acceder a las propiedades del objeto.

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 comprendemos que los objetos son una representación de entidades, podemos crear templates (plantillas) que contengan los datos de un objeto en particular.

Por ejemplo, si tuviéramos que desarrollar un portal de noticias sobre la industria automotriz, los artículos pueden ser generados a partir de plantillas basadas en la información almacenada en un objeto.

Supongamos que tenemos una sección HTML dedicada a los artículos que deseamos crear:

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

Podemos utilizar un objeto 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...`
};
```

Luego generamos el template del artículo con los datos del objeto y posteriormente lo agregamos 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;
}
```

Juntando todas las partes, el código de nuestro artículo quedaría así:

```
<!DOCTYPE html>
<html lang="es">
 <head>
   <meta charset="UTF-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Articulo</title>
   <style>
    .articulo {
```

```
      width: 250px;
      padding: 10px;
      background: lightgray;
     }
   </style>
 </head>
 <body>
   <section class="articulos"></section>
   <script>
     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...`
     };

    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>
    `;
   </script>
 </body>
</html>
```

Con esto, obtendremos un artículo con contenido dinámico y generado a partir de los datos de un objeto:

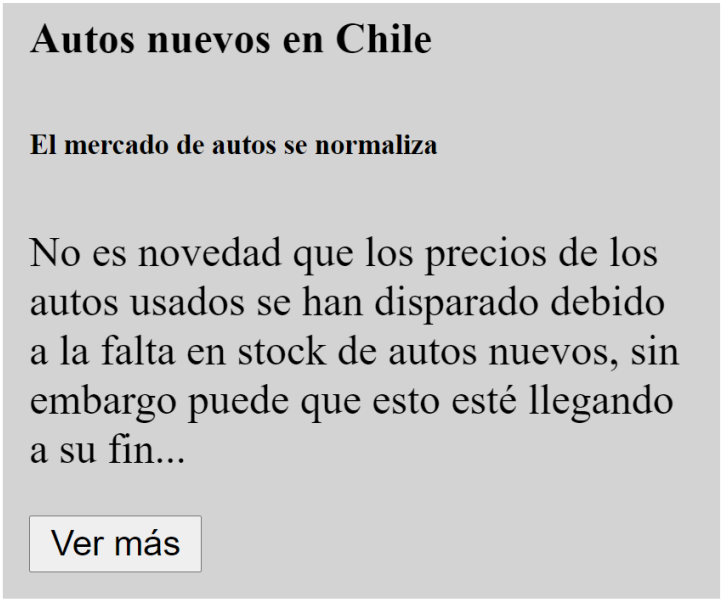

Imagen 3. Artículo generado con los datos de un objeto
Fuente: Desafío Latam

Actividad 7 : 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 que sabemos cómo representar las características de una entidad en un objeto y cómo acumular datos en arreglos, podemos dar el siguiente paso y combinar ambos tipos de datos en un ejercicio práctico. Imaginemos que necesitamos 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},
]
```

A partir de este arreglo podemos mostrarle al usuario una lista, interpolando los datos de cada venta, tal como se muestra a continuación:

```
<!DOCTYPE html>
<html lang="es">
 <head>
   <meta charset="UTF-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Tarjeta de Producto</title>
 </head>
 <body>
  <h3>Ventas de la semana:</h3>
  <ul></ul>
  <script>
    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},
    ]
```

```
    const ul = document.querySelector("ul")
    for(let venta of ventasDeLaSemana){
      ul.innerHTML+= `<li>${venta.dia}: ${venta.total} </li>`
    }
  </script>
 </body>
</html>
```

Esto nos dará como resultado lo siguiente:

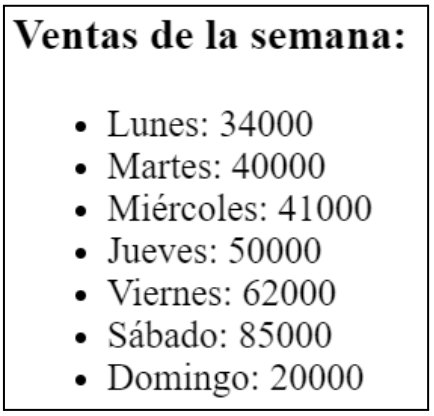

Imagen 4. Ventas de la semana
Fuente: Desafío Latam

Llenado de una tabla HTML

Sigamos practicando la utilidad de los arreglos de objetos mediante la generación de templates para llenar los registros de una tabla HTML que muestra los departamentos y los datos de contacto de un edificio residencial. Para esto ocuparemos el siguiente código:

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

El objetivo es agregar filas con la etiqueta “tr” al cuerpo de la tabla utilizando los datos proporcionados en el 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 lograrlo, 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;
}
```

El resultado será 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 son colecciones de datos que nos ayudan a agrupar información
- Una buena práctica para asignar 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 un 1. Con esto 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?

Solucionario Actividades

Solución Actividad 1

```
console.log(nombres[0]);
console.log(nombres[5]);
console.log(nombres[nombres.length - 1]);
```

Solución Actividad 2

```
<!DOCTYPE html>
<html lang="es">
 <head>
   <meta charset="UTF-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Nombres</title>
 </head>
 <body>
   <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>
 </body>
</html>
```

Solución Actividad 2.1

```
<!DOCTYPE html>
<html lang="es">
 <head>
   <meta charset="UTF-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Precios</title>
 </head>
 <body>
```

```
   <section id="precios"></section>
   <script>
      const precios = [1000, 2500, 3100, 4800, 7500]
      const preciosSection = document.getElementById("precios")
      for (let precio of precios){
         preciosSection.innerHTML += “<p>” + precio + “</p>”
      }
   </script>
 </body>
</html>
```

Solución Actividad 3

```
<!DOCTYPE html>
<html lang="es">
 <head>
   <meta charset="UTF-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Precios Optimizado</title>
 </head>
 <body>
   <section id="precios"></section>
   <script>
      const precios = [1000, 2500, 3100, 4800, 7500]
      const preciosSection = document.getElementById("precios")
      let html = “”
      for (let precio of precios){
        html += “<p>” + precio + “</p>”
      }
      preciosSection.innerHTML = html
   </script>
 </body>
</html>
```

Solución Actividad 4

```
<!DOCTYPE html>
<html lang="es">
 <head>
   <meta charset="UTF-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Mis datos</title>
 </head>
 <body>
   <ul id="mis-datos"> </ul>
 <script>
   const datos = [1200, 350, 1500, 1400, 250, 5000, 1950, 1952]
   const d = document.querySelector("#mis-datos")
   for (let valor of datos){
     const item = document.createElement(“li”)
     item.textContent = valor
     if (valor > 1000 && valor < 2000){
       item.style.color = ‘green’;
     } else {
       item.style.color = ‘red’;
     }
     d.appendChild(item)
   }
 </script>
 </body>
</html>
```

Solución Actividad 5

```
const guitarra = {
  cuerdas: 6,
  color: "Azul",
  año: "1994"
}

const computadora = {
  marca: "ASUS",
  ram: "16GB",
  so: "Windows",
  tipo: "Gamer"
}
```

```
const casa = {
  cuartos: 4,
  baño: 2,
  patio: 1,
  garage: 1,
  ubicación: "San Diego 8081"
}
```

Solución Actividad 6

```
console.log(persona["añoDeNacimiento"]); //
console.log(persona.añoDeNacimiento);
console.log(persona["apellido"]);// console.log(persona.apellido);
```

Solución Actividad 7

```
<!DOCTYPE html>
<html lang="es">
 <head>
   <meta charset="UTF-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Tarjeta de Producto</title>
   <style>
    .producto {
      width: 400px;
      padding: 30px;
      background: white;
      border : solid 1px black;
      text-align: center;
     }
     .producto img {
      width: 100%;
      max-width: 200px;
      height: auto;
     }
   </style>
 </head>
 <body>
   <section class="producto"></section>
   <script>
     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`
    };

    const productoSection = document.querySelector(".producto")

    productoSection.innerHTML = `
     <article class="articulo">
        <h4>${producto.titulo}</h4>
        <h6>Precio: $${producto.precio}</h6>
        <p>Color: ${producto.color}</p>
        <img src = "${producto.src}">
        <p>${producto.descripción}</p>
     </article>
    `;
   </script>
 </body>
</html>
```

Solución Actividad 8

```
<!DOCTYPE html>
<html lang="es">
 <head>
   <meta charset="UTF-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Galería de Iconos</title>
    <style>
    .iconos {
      display: grid;
      grid-template-columns: repeat(3, 1fr);
      grid-gap: 10px;
      text-align: center;
     }
     .icono {
      font-size: 50px;
      background-color: lightgray;
     }
     h1 {
```

```
      text-align: center;
    }
  </style>

</head>
<body>
  <h1> Iconos Divertidos </h1>
  <section class="iconos"></section>
  <script>
    const iconos = [
      { icono: "🚗", descripcion: "Auto" },
      { icono: "🤖", descripcion: "Robot" },
      { icono: "👻", descripcion: "Fantasma" },
      { icono: "👽", descripcion: "Alien" },
      { icono: "🦷", descripcion: "Diente" },
      { icono: "🕹️", descripcion: "Joystick" }
    ];
     const iconosSection = document.querySelector(".iconos")
     for (let icono of iconos){
       const template = `
         <div class = 'icono'>
           <div>${icono.icono}</div>
           <div>${icono.descripcion}</div>
         </div>
       `;
       iconosSection.innerHTML += template;
     }

   </script>
</body>
</html>
```