UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Diplomatura en programación web full stack con React JS

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 2: JavaScript

Unidad 4: Funciones anónimas y funciones arrow

Presentación:

Hasta ahora estuvimos trabajando con funciones que definimos con la palabra clave function seguida del nombre de la función (porque las funciones dijimos que tenían un nombre). En esta unidad vamos a aprender 2 nuevas formas de definir funciones que son las funciones anónimas (no tienen nombre) y las funciones arrow (llevan una flecha! =>). Son muy útiles y la documentación de JavaScript está llena de referencias a este tipo de funciones por lo que aprenderlas es sumamente útil.

Objetivos:

Que los participantes:

- Sean capaces de identificar las funciones anónimas y las funciones arrow.
- Sepan crear funciones anónimas y funciones arrow.

Bloques temáticos:

1. Funciones anónimas
2. Funciones Arrow
3. Ejemplo de funciones anónimas
4. Ejemplo de funciones arrow
5. Trabajo Práctico

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

** El MEC es el modelo de E-learning colaborativo de nuestro Centro.*

Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Funciones anónimas

En JavaScript es muy habitual trabajar con funciones, y hasta pasar una función como parámetro a otra función. Un ejemplo de ellos es la función setInterval, la cual ejecuta una función cada X milisegundos.

La función setInterval tiene el siguiente prototipo

setInterval(<función a llamar cada X milisegundos>, <X milisegundos);

Ejemplo

```
function mostrarFechaHora() {
      console.log(new Date());
}
setInterval(mostrarFechaHora, 1000);
```

Esto llamará a la función mostrarFechaHora cada 1 segundo (1000 milisegundos)

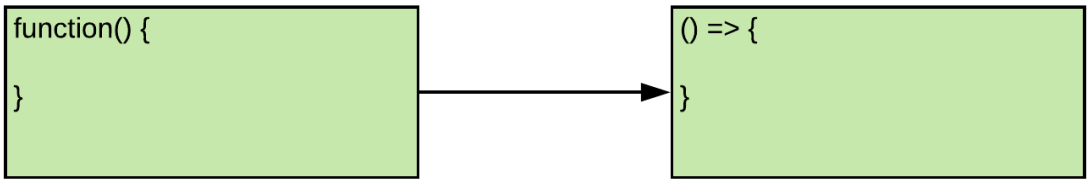
Ejemplo

Otra forma de escribir el mismo código es la siguiente:

```
setInterval(function() {
      console.log(new Date());
}, 1000);
```

En vez de definir una función mostarFechaHora, se incluye el código de la función directamente en el llamado de setInterval(). Al pasar la función, estamos realizando el mismo proceso que el definir una función con un nombre y pasarla como parámetro. Es importante destacar que al pasar una función directamente como parámetro, no se incluye nombre de la misma setInterval(function()...). Este tipo de funciones, son llamadas funciones anónimas.

A partir de la versión 6 de EcmaScript (definición de JavaScript) se incluye una nueva forma de pasar como parámetro estas funciones anónimas. La llamada arrow function.



2. Funciones Arrow

Se sustituye **function() {}** por **()=>{}**

Tomando como referencia el código anterior, convertido a función arrow, el mismo queda de la siguiente forma

Ejemplo

```
setInterval(() => {
      console.log(new Date());
}, 1000);
```

Este tipo de funciones tiene algunas ventajas que iremos viendo a lo largo del curso.

3. Ejemplo de funciones anónimas

Recorrer vectores

Disponemos de diferentes alternativas para recorrer vectores en JavaScript, una de las más utilizadas es forEach. La operación forEach es un método de todos los vectores, y por lo tanto podemos utilizarlo para cualquier vector que utilicemos. El método forEach recibe como parámetro una función que será llamada una vez por cada elemento del vector (pasándole como parámetro el valor del elemento).

Ejemplo

```
var edades = [10,33,12,74,22,84,44];
edades.forEach(function(unElemento) {
    console.log(unElemento);
});
```

La salida por pantalla del código será

10
33
12
74
22
84
44

Llamándose a la función function(unElemento) {} una vez por cada elemento que exista en el vector. Para el ejemplo mostrado, la función será llamada 7 veces.

4. Ejemplo de funciones arrow

Encontrar la posición de un elemento en el vector (findIndex)

Muchas veces deseamos encontrar la posición de un elemento dentro de un vector solo teniendo una propiedad del elemento que estamos buscando (no todo el elemento completo). Por ejemplo, si tenemos un listado de personas, puede ser que solo tengamos el ID de la persona que deseamos encontrar, pero no todos los datos de la misma. En estos casos, podemos utilizar el método findIndex() cuyo prototipo es el siguiente

```
<vector>.findIndex(item => {
      return <condición que se debe cumplir para haber encontrado el elemento>
})
```

Supongamos que tenemos el siguiente vector

```
var tareas = [
                {
                    "id": 1,
                    "title": "delectus aut autem",
                },
                {
                    "id": 2,
                    "title": "quis ut nam facilis et officia qui",
                },
                {
                    "id": 3,
                    "title": "fugiat veniam minus",
                }
            ]
```

Y deseamos encontrar la posición de la tarea cuyo id es igual a 2. Utilizando findIndex llegaríamos al siguiente código

```
var posicion = respuesta.findIndex(item => {
```

```
    return item.id==2
})
console.log(posicion);
```

Para este caso, posición tiene el valor 1, ya que es el segundo elemento del vector.

Eliminar elementos del vector (splice)

Cuando deseamos reducir un vector a menos cantidad de elementos, por ejemplo, si tenemos 10 elementos, pero sólo deseamos quedarnos con los primeros 5 elementos, podemos usar el método splice. El cual tiene el siguiente prototipo

```
<vector>.splice(<desde>);
<vector>.splice(<desde>, <cantidad>);
```

Para el primer caso, splice elimina del vector los elementos desde la posición indicada <desde> hasta la finalización del vector.

Para el segundo caso, splice elimina los elementos desde la posición <desde> y desde esa posición eliminará <cantidad> elementos.

Adicionalmente el método splice puede ser utilizado para agregar un elemento al vector, en una posición determinada. Para ello, utilizamos la siguiente sintaxis

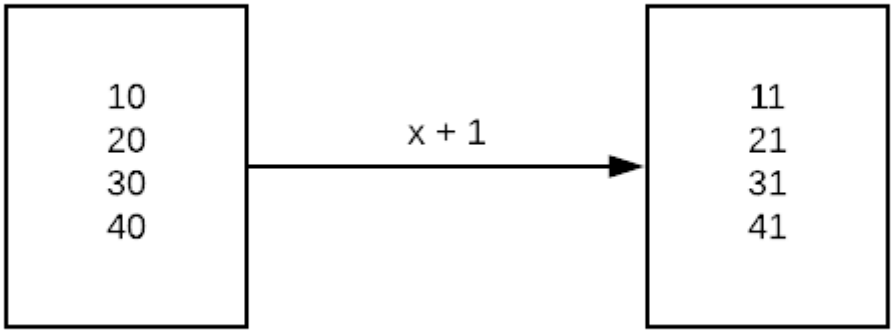
```
<vector>.splice(<posición>, 0, <nuevo elemento>);
```

En la posición <posición> se agrega <nuevo elemento>

Convertir un vector (map)

Si deseamos convertir los elementos de un vector, aplicarle alguna transformación, para ello disponemos del método **map**. El cual permite convertir una estructura A al formato de

la estructura B. Si queremos sumar 1 a todos los valores que tenemos en un vector origen, podemos realizar una función map que realice la transformación.



Ejemplo

```
var edades = [10,33,12,74,22,84,44];
edades = edades.map(unItem => {
     return unItem+1;
})

console.log(edades);
```

La salida será

```
[ 11, 34, 13, 75, 23, 85, 45 ]
```

La función map, es especialmente útil cuando deseamos modificar el contenido de un vector JSON de objetos (JSON) a otro formato. Si deseamos convertir el vector de personas al formato {nombre_completo: ‘’, edad: 0} debemos aplicarle una transformación al vector original

Ejemplo

```
var personas = [
      { nombre: 'Juan', apellido: 'Perez', edad: 40},
      { nombre: 'Maria', apellido: 'Gonzalez', edad: 33 }
]

personas = personas.map(unItem => {
      return {
            nombre_completo: unItem.nombre + ' '+unItem.apellido,
            edad: unItem.edad

      }

})

console.log(personas);
```

Para este caso, la salida por pantalla será

```
[
      { nombre_completo: 'Juan Perez', edad: 40 },
      { nombre_completo: 'Maria Gonzalez', edad: 33 }
]
```

5. Trabajo Práctico

Modificar el carrito desarrollado en la Unidad 2 para utilizar funciones arrow.

Bibliografía utilizada y sugerida

MDN - JavaScript. (n.d.) Recuperado de: https://developer.mozilla.org/es/docs/Web/JavaScript

Udemy. (n.d.) Recuperado de: https://udemy.com

Wikipedia - JavaScript. (n.d.) Recuperado de https://es.wikipedia.org/wiki/JavaScript

w3schools.com - JavaScript. (n.d.) Recuperado de https://www.w3schools.com/js/

Lo que vimos:

- Funciones anónimas y funciones arrow.

Lo que viene:

- Módulo 3: NodeJS Introducción.