Práctica de San Cristobal Seguros - Frontend

Libro de Laboratorios

November 9, 2023

Acerca de este documento

Este documento se genero a partir de fuentes en LaTeX en https://github.com/bootlin/training-materials.

Correcciones, sugerencias, contribuciones son bienvenidas!

Ejercicios adicionales

Ejercicios

1. Implemente su propia función `map`
2. Implemente su propia función `find`
3. Implemente su propia función `filter`
4. Implemente su propia función `reduce`
5. Queremos realizar cálculos usando funciones y obtener los resultados. Los requerimientos son:
   - Debe haber una función para cada número del `0` ("*cero*") al `9` ("*nueve*")
   - Debe haber una función para cada una de las siguientes operaciones matemáticas: `suma`, `resta`, `multiplicación` y `división`
   - Cada cálculo consta exactamente de una operación y dos argumentos (números)
   - La función externa representa el operando izquierdo, la función interna representa el operando derecho
   - La división debe ser una división entera. Por ejemplo, el resultado debería ser `2` y no `2.666666...`

```
seven(times(five()));  // debe devolver 35
four(plus(nine()));    // debe devolver 13
eight(minus(three())); // debe devolver 5
six(dividedBy(two())); // debe devolver 3
```


TypeScript - Clases

Ejercicio

Crear una clase que represente una pila (LIFO). Dicha clase debe tener los siguientes métodos:

- Agregar elementos a la pila `push`
- Sacar elementos de la pila `pop`
- Consultar la cantidad de elementos `size`

Además:

1. Diseñar la clase de forma tal que se puedan crear pilas para tipos concretos, ej: boolean, string, number, o cualquier otro tipo. Pero que solo permita dichos tipos.
2. Crear una nueva clase que no permita el agregado de elementos repetidos.

Ejercicio

1. Crear una clase para representar a las figuras geométricas, que permita calcular el `perimetro` y el `area`
2. Crear una clase que represente a los cuadrados.
3. Crear una clase que represente a los círculos.
4. Aplicando polimorfismo, muestre un ejemplo utilizando las clases anteriores.

Creando y Comunicando Componentes entre si

Componentes

Vamos a armar la aplicacion de TATETI. Para ello vamos a necesitar los siguientes componentes:

- Un componente que represente la grilla del juego de 9x9
- Un componente que represente a la celda donde puede haber una `X` o un `O`

El estado de las celdas las mantiene el componente grilla el cual le pasa el estado al componente celda para que visualice un `X` o `O`.

Al asignar un estado a la celda, el componente debe avisar a la grilla del cambio.

Si selecciono por primera vez una de las celdas se cambiara el estado a `X` o `O` solamente si no habia sido seleccionada antes.

La secuencia de `X` y `O` se da de forma alternada.

Crer un formulario reactivo

Ejercicio

Armar un formulario reactivo de subscripción a una revista. EL mismo debe tener los siguientes controles:

- Nombre
- Apellido
- Email
- Confirmación del email
- Telefono
- Contraseña
- Si quiere recibir notificaciones de la página
- Confirmación de los terminos y condiciones

Los campos son requeridos y se deben validar de acuerdo al tipo.

El email y la confirmación del email debe ser iguales.

Utilizar `FormBuilder` para construir el formulario.

Al enviarlo, lo debe recibir un servicio el cual va a visualizar el objeto de la subscripción por consola.

Conexión con el Backend

Ejercicio

Dado el formulario anterior, de suscripción a una revista, vamos a crer un servicio que permita conectarnos al Backend con los siguientes métodos:

- `crearSuscripcion`
- `buscarSuscripcionPorId`
- `buscarSuscripciones`
- `borrarUnaSuscripcion`

Adapte el componente de suscripción anterior para que utilice el servicio.

