Guia de Ejercicios

Clase IV

Multiple Choice

1. ¿Qué método http debo usar para una llamada para obtener algún recurso?
    a. GET
    b. POST
    c. DELETE
    d. OPTIONS
    e. PUT
2. El código http para indicar una respuesta estándar de éxito en la comunicación http es:
    a. 200 OK
    b. 201 OK
    c. 204 OK
    d. 200 Contenido Parcial
    e. 201 Creado
    f. 206 No hay contenido
    g. 204 OK
3. Una forma simple de obtener una colección de datos, a partir de otra obtenida previamente es con el método _____ de la colección:
    a. map
    b. reduce
    c. for
    d. forEach
    e. while

4. Tengo un arreglo de jugadores, donde el tipo de datos Jugador tiene el siguiente formato:

```
enum PiernaAgil { Izquierda, Derecha, Ambas, Ninguna};

interface Jugador {
      edad: number;
      piernaAgil: PiernaAgil;
      nombre: string;
      velocidad: string;
            ...
}
```

Queremos obtener, de esa colección, el conjunto de jugadores menores de 27 años. Para ello, la forma correcta de hacerlo es:

a.

```
jugadores.filter( jugador => jugador.edad < 27);
```

b.

```
jugadores.filter( jugador => {
    if( jugador.edad < 27 ) {
        return jugador;
    } else {
        return null;
    }
}
```

c.

```
jugadores.map( jugador => jugador.edad < 27 )
```

d.

```
jugadores.map( jugador => {
      if( jugador.edad < 27 ) {
            return jugador;
      } else {
          return null;
      }
}
```

e.

```
for( let jugador of jugadores ) {
      if( jugador.edad < 27 ) {
      return jugador;
```

```
}
}
```

5. Teniendo un arreglo de números, los cuales representan distintas alturas (ej: let alturas = [3, 27.5, ...]), una forma simple de obtener la altura promedio sería:

    a.

```
alturas.reduce( (alturaPromedio, altura, _, alturas) => alturaPromedio + altura/alturas.length, 0);
```

    b.

```
alturas.mean();
```

    c.

```
alturas.map(altura => altura/alturas.length);
```

    d.

```
alturas.reduce( (alturas, altura) => altura/alturas.length);
```

    e.

```
alturas.filter( altura => altura/alturas.length);
```

Ejercicios de elaborar

1. En la guia anterior, desarrollamos la interface FuncionGenerica:

```
interface FuncionGenerica<Recibe, Devuelve> {
      evaluar: (x: Recibe) => Devuelve;
}
```

    Recibe es un tipo de datos, mientras que *Devuelve* puede ser otro tipo de datos distinto de *Recibe*.

    Las funciones que implementen a la función genérica, pueden trabajar en distintos campos. Podemos volver a trabajar con simples números, como arreglos, matrices, listas, objetos.

    Crear implementaciones de FuncionGenerica que trabaje con listas de numeros, y apliquen un mapeo de estos numeros a las aplicaciones de Funcion que se usaron en las clases (usar el método map visto en la clase IV).

Ejemplos aplicando una FuncionGenerica de arreglos:

```
const sinArreglo = new TrigonometricaArreglos(
FuncionesTrigonometricas.SIN)’
sinArreglo.evaluar([0, pi/4, pi/2, 3pi/4, pi]); // [0, raiz(2)/2 , 1 ,
0]
const funcionCuadraticaSimpleArreglo = new PolinomicaArreglos([0, 0,
1]); // x^2
funcionCuadraticaSimpleArreglo.evaluar([2,3,4,5]) = [4, 9, 16, 25]
```

2. En la física, hay una medida que permite poner en números la propiedad que tienen los cuerpos de permanecer en estado reposo o movimiento relativos. A esta medida se la llama “momento de inercia”.
   El momento de inercia refleja la distribución de masa de un cuerpo o de un sistema de partículas en rotación, respecto a un eje de giro. El momento de inercia solo depende de la geometría del cuerpo y de la posición del eje de giro; pero no depende de las fuerzas que intervienen en el movimiento.
   Para calcular el momento de inercia de un cuerpo, o sistema de partículas, se tiene la siguiente ecuación, donde mi es la masa de la partícula sub i y ri es la distancia al eje de la partícula sub i:

$$I = \sum_i m_i r_i^2$$

   Nuestro modelado de las partículas (y del cálculo de la inercia de cada masa con respecto al origen) es el siguiente:

```
class Particula{
    private x: number;
    private y: number;
    private z: number;
    private masa: number;

    constructor( x: number, y: number, z: number, masa: number) {
        this.x = x;
        this.y = y;
        this.z = z;
        this.masa = masa;
}

    calcularInercia() : number {
        return (this.x ** 2 + this.y ** 2 + this.z ** 2)*this.masa;
```

```
    }
}
```

Se pide realizar una función momentoInerciaCuerpo que, dado un cuerpo (arreglo de partículas), calcule el momento de inercia de ese cuerpo.

Un ejemplo sería si tenemos el siguiente cuerpo:

```
const cuerpo: readonly Particula[] = [new Particula(0, 0, 0, 200), new
Particula(100, 100, 100, 0), new Particula(10, 10, 10, 200)];
```

La inercia de este cuerpo se deberia poder calcular como:

```
momentoInerciaCuerpo(cuerpo); // 60000
```

Para ello, se deben utilizar los métodos map y reduce, aprendidos en esta clase.

Una posible solución es la siguiente:

```
function momentoInerciaCuerpo (cuerpo: readonly Particula[]) {
    return cuerpo.map( particula => particula.calcularInercia() )
                 .reduce( (momentoDeInercia: number, inercia: number)
=> momentoDeInercia + inercia, 0);
}
```

3. Realizar un programa de consultas (API) con Express. El mismo tiene que ubicarse en el puerto 4567.
   Los datos sobre los que se harán consultas están representados por una Partícula, como el ejercicio anterior, pero con más información:

   Link a Particula

   POST /particula

   Crea una partícula con los parámetros pasados en este formato JSON:

   {

   posicionInicial: {

   x: number;

   y: number;

   z: number;

```
        }
        velocidadInicial: {
                x: number;
                y: number;
                z: number;
        }
        masa: number;
}
```

GET /particula/inercia

Obtiene el momento de inercia de la particula

GET /particula/momento

Obtiene el momento de la partícula

GET /particula/energia

Obtiene la energía cinética de la partícula en ese momento

PUT /particula/velocidad

Modifica la velocidad de la partícula. Como parámetros se pasan la velocidad y el tiempo de aceleración en este formato:

```
{
        nuevaVelocidad: {
                x: number;
                y: number;
                z: number;
        }
        tiempoDeAceleracion: number;
```

}

Lo que devuelve la modificación es la aceleración que tiene que tener la partícula para llegar a la velocidad acordada en el tiempo acordado.

PUT /particula/posicion

Modifica la posición de la partícula. Como parámetros se pasa únicamente la nueva posición de la partícula:

```
{
        nuevaPosicion: {
                                x: number;
                                y: number;
                                z: number;
        }
}
```

Lo que devuelve la modificación de la posición es el tiempo que tarda la partícula en moverse hacia la posición acordada. Tener en cuenta que el tiempo se cuenta en distancia Manhattan (se suma el tiempo que tarda en cada eje).

¿Qué pasa si la velocidad en un eje donde se tiene que mover es 0?

Agregar todas las funciones, clases, interfaces, métodos que precisen necesarios para el correcto funcionamiento del programa