Guia de Ejercicios

Clase III

Multiple Choice

1. ¿Cuál es el propósito del uso de Generics en TS?
    a. Proveer una manera de definir clases y funciones que puedan trabajar con cualquier tipo de datos
    b. Restringir el tipo de datos con las que puede trabajar cierta clase o función
    c. Proveer una manera de realizar operaciones aritméticas en tipos de datos
2. ¿Cual es un ejemplo de una funcion generica en TypeScript?
    a. function sumar(a: number, b: number): number {

        return a+b;

        }
    b. function revertirString(str: string): string {

        return str.split(“”).reverse().join(“”);

        }
    c. function identidad<T>(arg: T) {

        return arg;

        }
3. ¿Cual es la sintaxis para definir un parámetro de tipo genérico en TS?
    a. <T>
    b. [T]
    c. (T)
4. ¿Qué palabra se utiliza en TypeScript para arrojar excepciones?
    a. throw
    b. Error
    c. return
5. Danfojs basa gran parte de su comportamiento en base al objeto:
    a. DataFrame
    b. DataSet

c. Object

6. Si tengo el siguiente código:

```
df.groupby(['DEALSIZE']).agg({'PRICEEACH': 'max'})
```

¿Qué se estaría queriendo calcular del DataFrame df en este caso?

a. Obtener el máximo precio por unidad en total

b. Le agrega una nueva columna al dataset, llamada max

c. Obtener todos los precios por unidad de cada DEALSIZE

d. Arroja un error de compilación

e. Obtener el máximo precio por unidad de cada DEALSIZE

¿Qué es la recursividad en TypeScript?

a) Un método para crear una instancia de una clase en TypeScript

b) Una forma de definir múltiples métodos dentro de una sola función en TypeScript

c) Una técnica para dividir un problema en subproblemas más pequeños y resolverlos de manera recursiva

¿Cuál es la forma más común de implementar la recursividad en TypeScript?

a) Usar bucles for o while

b) Usar una función que se llama a sí misma

c) Usar una variable global para realizar el seguimiento del estado

¿Cuál es la diferencia entre una función recursiva y una función iterativa en TypeScript?

a) Las funciones recursivas son más lentas que las iterativas.

b) Las funciones iterativas son más difíciles de entender que las recursivas.

c) Las funciones recursivas se llaman a sí mismas y las iterativas usan bucles.

¿Qué es la condición de salida en una función recursiva?

a) El valor que se devuelve al final de la función.

b) La condición que indica cuándo la función debe detenerse.

c) La cantidad de veces que la función se llama a sí misma.

¿Por qué es importante tener una condición de salida en una función recursiva en TypeScript?

a) Para evitar errores de tiempo de ejecución.

b) Para garantizar que la función se ejecute al menos una vez.

c) Para evitar un bucle infinito y un desbordamiento de la pila.

Ejercicios de elaborar

Para implementar una interfaz, función, clase, etc que recibe múltiples tipos genéricos, se ponen dentro de los brackets (<>), separados por comas, los distintos tipos genéricos. Un ejemplo puede ser con la interfaz Funcion. Si quisieramos realizar una FuncionGenerica, esta sería del tipo:

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

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.

1. 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.

   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, 1])
funcionCuadraticaSimpleArreglo.evaluar([2,3,4,5]) = [4, 9, 16, 25]
```

2. Crear *implementaciones* de FuncionGenerica que reciban un objeto de Coordenadas (x,y,z) y devuelvan estas coordenadas mapeadas aplicando las funciones utilizadas en la clase.

```
class Coordenadas {
      x: number;
      y: number;
      z: number;

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

Ejemplos aplicando una FuncionGenerica de Coordenadas:

```
const coordenadas1 = new Coordenadas(pi/2, pi/4, 0);
sinCoordenadas.evaluar(coordenadas1) = { x: 1, y: raiz(2)/2, z:  0 }
const coordenadas2 = new Coordenadas(1, 2, 3);
funcionCuadraticaSimpleCoordenadas.evaluar(coordenadas2) = { x: 1, y: 4,
z: 9 }
```

3. Crear una FuncionGenerica que tome un string y devuelva la cantidad de vocales incluidas dentro del string (siendo a, e, i, o, u, sin tener en cuenta tildes).

4. Crear *implementaciones* de FuncionGenerica, para poder trabajar con un argumento de 2 binarios (el binario izquierdo, y el binario derecho).

```
interface DosBinarios {
      izquierdo: boolean;
      derecho: boolean;
}
```

Con el And, si alguno de los valores es falso, el resultado es falso. Sino, es verdadero:

```
And.evaluar({izquierdo: true, derecho: false}) = false
```

Con el Or, si alguno de los valores es verdadero, el resultado es verdadero. Sino, es falso:

```
Or.evaluar({izquierdo: true, derecho: false}) = true
```

Con el Xor, si todos los valores son verdaderos (si el AND es verdadero), el resultado es falso. Si todos los valores son falsos (si el OR es falso), el resultado es falso . Sino, el resultado es verdadero.

```
Xor.evaluar({izquierdo: true, derecho: false}) = true
Xor.evaluar({izquierdo: false, derecho: false}) = false
Xor.evaluar({izquierdo: true, derecho: true}) = false
Xor.evaluar({izquierdo: false, derecho: true}) = true
```

5. Crear una función genérica que toma un arreglo de cualquier tipo y retorna el primer elemento del arreglo. La función debería funcionar con arreglos de cualquier tipo y retornar el primer elemento del mismo. En el caso de no haber elementos dentro del arreglo, retornar *undefined*.

```
const strArr = ["a", "b", "c"];
console.log(primerElemento(strArr)); // "a"
const numArr = [1, 2, 3];
console.log(primerElemento(numArr)); // 1
const arrVacio = [];
console.log(primerElemento(arrVacio)) // undefined
```

6. Crear una clase abstracta Vehiculo, con las propiedades marca, modelo y año. Extender la misma con una clase Auto, que tiene la propiedad numDoors.
7. Crear una clase Persona, que tiene las propiedades nombre y edad. También, tendra una propiedad rol, que es del tipo Rol, una clase abstracta con un método obtenerDescripcion. De la misma extendera Estudiante, que tiene las propiedades escuela y promedio. en obtenerDescripcion, tiene que mostrar un string como el siguiente: “El estudiante va a la escuela “ seguido del nombre de la escuela, seguido por “ y tiene el promedio “ seguido del promedio.
8. Crear una clase abstracta Figura, con propiedades de ancho y largo, y con un método abstracto *getArea()*. Extender la misma con una clase Rectangulo, y definir el método *getArea()*, que debería devolver el área del rectángulo.
9. Extender la clase abstracta Figura con otras clases Circulo, Triangulo y Cuadrado y definir el método *getArea(), que debería devolver el área de cada figura.*

Excepciones

1. Escribir una función que tome dos números como argumento y devuelva el resultado de dividir el primero por el segundo. Usar la secuencia de try catch

para solucionar el problema de que el segundo número sea 0. Imprimir a consola que no se puede dividir por 0.

2. Escribir una función que tome un número y calcule su raíz cuadrada. Usar la secuencia de try catch para solucionar el problema de que ese mismo número sea menor que 0