TypeScript - Functiones

Objetivos: Convertir las funciones existentes de JavaScript a TypeScript, Entender las funciones como tipos, Convierte funciones específicamente tipificadas a funciones más flexibles genéricas

Ejercicio 3.1

Dado el siguiente codigo:

```
function add(x, y) {
  return x + y;
}

function sumArray(numbers) {
  return numbers.reduce(add, 0);
}

const someSum = sumArray(['3', '6', '9']);

console.log('[Ejercicio 3.1]', `3 + 6 + 9 === ${someSum}`);
```

1 Agregue tipos explícitos a los parámetros y el tipo de retorno

2 Solucione cualquier error resultante de tipos inválidos

Ejercicio 3.2

Dado el siguiente codigo:

```
const bankAccount = {
  money: 0,
  deposit(value, message) {
    this.money += value;
    if (message) {
      console.log(message);
    }
  }
};

bankAccount.deposit(20);
bankAccount.deposit(10, 'Deposit received')

console.log('[Exercise 3.2]', `Account value: $${bankAccount.money}`);
```

1 Agregue tipos explícitos a los parámetros y el tipo de retorno a la funcion ‘deposit‘

2 Haz que el parámetro de ‘message‘ sea *optional*

Ejercicio 3.3

Para una palabra dada, calculamos su puntuación en Scrabble®

```
function computeScore(word) {
  const letters = word.toUpperCase().split('');
  return letters.reduce((accum, curr) => accum += getPointsFor(curr), 0);
}

function getPointsFor(letter) {
  const lettersAndPoints = [
    ['AEOIULNRST', 1],
    ['DG', 2],
    ['BCMP', 3],
    ['FHVWY', 4],
    ['K', 5],
    ['JX', 8],
    ['QZ', 10],
  ];

  return lettersAndPoints.reduce((computedScore, pointsTuple) => {
    const [letters, score] = pointsTuple;
    if (letters.split('').find((ll) => ll === letter)) {
      return computedScore += score;
    }
    return computedScore;
  }, 0);
}

console.log('[Ejercicio 3.3]', `zoologico vale ${computeScore('zoo')} puntos.`);
```

1 Añadir anotaciones de tipo siempre que sea posible

Ejercicio 3.4

Dado el siguiente codigo:

```
function greet(greeting) {
  return greeting.toUpperCase();
}

const defaultGreeting = greet();
const portugueseGreeting = greet('Oi como vai!');

console.log('[Ejercicio 3.4]', defaultGreeting, portugueseGreeting);
```

1 Añadir tipos explícitos a los parámetros y tipo de retorno

2 Añadir un saludo predeterminado: ”hola”

Ejercicio 3.5

```
function layEggs(quantity, color) {
```



```
  console.log(
    `[Ejercicio 3.5] Acabas de poner ${quantity} huevos ${color}. Buen trabajo!`);
}

layEggs();
```

1 Añadir anotación de tipo de parámetro

2 A pesar de que esta función no vuelve, agregue un tipo de retorno explícito

Ejercicio 3.6

Aquí hemos inicializado dos variables con tipos de funciones. Posteriormente les asignamos funciones.

```
let multiply: (val1: number, val2: number) => number;
let capitalize: (val: string) => string;

multiply = function (value: string): string {
  return `${value.charAt(0).toUpperCase()}${value.slice(1)}`;
}

capitalize = function (x: number, y: number): number {
  return x * y;
}

console.log('[Ejercicio 3.6]', capitalize(`habil ${multiply(5, 10)}`));
```

1 Arreglar los errores

Ejercicio 3.7

Actualmente, nuestra función 'pushToCollection' acepta *cualquier* elemento y lo agrega, (indiscriminadamente) a *cualquier* tipo de matriz.

Un par de problemas con esto:

1 El tipo 'any' hace que perdamos toda la información de tipos en nuestros parámetros.

2 Esta holgura se ha vuelto en nuestra contra durante el tiempo de ejecución (mira a 'incrementByTwo')

Dado el siguiente codigo:

```
const numberCollection: number[] = [];
const stringCollection: string[] = [];

function pushToCollection(item, collection) {
  collection.push(item);
  return collection;
}

// Anadir algunas cosas a las colecciones
pushToCollection(false, stringCollection);
pushToCollection('hi', stringCollection);
```



```
12 pushToCollection([], stringCollection);
13
14 pushToCollection('1', numberCollection);
15 pushToCollection('2', numberCollection);
16 pushToCollection('3', numberCollection);
17
18 const incrementedByTwo = numberCollection.map((num) => num + 2);
19
20 console.log('[Ejercicio 3.7]', `[${incrementedByTwo}] debe ser igual a [3,4,5]`);
```

1 Implementar 'pushToCollection' como una función genérica. (Esto debería crear errores en tiempo de compilación en lugares donde se agregan valores incorrectos a una colección determinada. Fije estos valores a los tipos correctos)

2 Una vez hecho genérico, 'pushToCollection' debe ser suficientemente *generic* para operar en artículos y colecciones de cualquier tipo mientras se continúa aplicando que conicidan

