TypeScript - Tipos

Objetivos: Comprender cómo TypeScript realiza análisis de flujo de código, Crear y aplicar tipos de unión e intersección, Utilice protecciones de tipo básico (tipos de estrechamiento con: typeof, instanceof, etc.)

Ejercicio 4.0

TypeScript es inteligente sobre los posibles tipos de una variable, dependiendo de la ruta del código.

Dado el siguiente codigo:

```
function doStuff(value: any): void {
  if (typeof value === 'string') {
    console.log(value.toUpperCase().split('').join(' '));
  } else if (typeof value === 'number') {
    console.log(value.toPrecision(5));
  }

  value; // coloque el cursor sobre `valor` aqui
}

doStuff(2);
doStuff(22);
doStuff(222);
doStuff('hello');
doStuff(true);
doStuff({});

console.log('[Ejercicio 4.1]');
```

1 Simplemente inspeccione los tipos posibles moviéndose sobre el 'texto' para ver cómo cambia el tipo inferido si se pueden hacer suposiciones de forma segura sobre los tipos posibles dentro de la ruta del código dado

Ejercicio 4.1

Dado el siguiente codigo:

```
interface EggLayer {
  layEggs(): void;
}

interface Flyer {
  fly(height: number): void;
}

```



```
interface Swimmer {
  swim(depth: number): void;
}

// agregar alias de tipo(s) aqui

class Bird implements BirdLike {
  constructor(public species: string) { }

  layEggs(): void {
    console.log('Poniendo huevos de aves.');
  }

  fly(height: number): void {
    console.log(`Volando a la altura de ${height} metros.`);
  }
};

class Fish implements FishLike {
  constructor(public species: string) { }

  layEggs(): void {
    console.log('Poniendo huevos de pescado.');
  }

  swim(depth: number): void {
    console.log(`Nadando a una profundidad de ${depth} metros.`);
  }
}

function getRandomAnimal() {
  const animals = [
    new Bird('puffin'),
    new Bird('kittiwake'),
    new Fish('sea robin'),
    new Fish('leafy seadragon'),
  ];

  return animals[Math.floor(Math.random() * animals.length)];
}

function interrogateAnimal(animal = getRandomAnimal()) {
  animal.swim(10) // se llama solo si es un pez
  animal.fly(10); // se llama solo si es un pajaro

  return animal.species;
}

console.log('[Ejercicio 4.4]',
  `Tenemos un ${interrogateAnimal()} en nuestras manos!`);
```

1 Restrinja el tipo de ‘valor‘ a ‘string o number‘

2 Solucione cualquier error resultante

Ejercicio 4.2

Dado el siguiente codigo:

```
1
2 function padLeft(value: string, padding: number | string): string {
3   // si padding es un numero, return `${Array(padding + 1).join(' ')}${value}`
4   // si padding es una cadena, return padding + value
5 }
6
7 console.log('[Ejercicio 4.2]', `
8       ${padLeft('', 0)}
9       ${padLeft('', '')}
10      ${padLeft('', '')}
11      ${padLeft('', '')}
12      ${padLeft('', '')}
13    `);
```

1 Use un protector de tipo para completar el cuerpo de la función ‘padLeft‘

Ejercicio 4.3

Dado el siguiente codigo:

```
1 const numbers = [1, 2, 3, [44, 55], 6, [77, 88], 9, 10];
2
3 function flatten(array) {
4   const flattened = [];
5
6   for (const element of array) {
7     if (Array.isArray(element)) {
8       element; // any[]
9       flattened.push(...element);
10    } else {
11      element; // any
12      flattened.push(element);
13    }
14  }
15
16  return flattened;
17 }
18
19 const flattenedNumbers = flatten(numbers);
20
21 console.log('[Ejercicio 4.3]', flattenedNumbers);
```

1 Añadir anotaciones de tipo (‘any‘ excluido)

2 Inspeccione el tipo de ‘element‘ en diferentes ramas de código

3 Bonificación: convertir ‘flatten‘ en una función genérica

Ejercicio 4.4

Dado el siguiente codigo:

```
interface EggLayer {
  layEggs(): void;
}

interface Flyer {
  fly(height: number): void;
}

interface Swimmer {
  swim(depth: number): void;
}

// agregar alias de tipo(s) aqui

class Bird implements BirdLike {
  constructor(public species: string) { }

  layEggs(): void {
    console.log('Poniendo huevos de aves.');
  }

  fly(height: number): void {
    console.log(`Volando a la altura de ${height} metros.`);
  }
};

class Fish implements FishLike {
  constructor(public species: string) { }

  layEggs(): void {
    console.log('Poniendo huevos de pescado.');
  }

  swim(depth: number): void {
    console.log(`Nadando a una profundidad de ${depth} metros.`);
  }
}

function getRandomAnimal() {
  const animals = [
    new Bird('puffin'),
    new Bird('kittiwake'),
    new Fish('sea robin'),
    new Fish('leafy seadragon'),
  ];

  return animals[Math.floor(Math.random() * animals.length)];
}

```



```
50 function interrogateAnimal(animal = getRandomAnimal()) {
51   animal.swim(10) // se llama solo si es un pez
52   animal.fly(10); // se llama solo si es un pajaro
53
54   return animal.species;
55 }
56
57 console.log('[Ejercicio 4.4]',
58   `Tenemos un ${interrogateAnimal()} en nuestras manos!`);
```

1 Las aves y los peces ponen huevos. Sólo los pájaros vuelan. Sólo los peces nadan. Defina dos tipos nuevos: 'BirdLike' y 'FishLike' basados en estos rasgos

2 Crea un alias de tipo para 'Bird OR Fish' llamado 'Animal'

3 Use 'instanceof' en 'interrogateAnimal' para permitir a los peces nadar y a los pájaros volar

4 Agregue cualquier anotación de tipo faltante, siendo lo más explícito posible

