TypeScript Arrays & Tuples

TypeScript Arrays

An **array** in TypeScript is a special type of variable that can store multiple values. These values can be of the same type or a combination of different types.

Declaring an Array

Arrays in TypeScript can be declared in two main ways:

- **Using square brackets ([])** (array literal syntax).
- **Using Array<Type>** (generic array syntax).

Approach 1: Array Literal

```
let names: string[] = [];  // Declaration of an empty string array

// Initializing values into the array

names[0] = "john";

names[1] = "smith";

names[2] = "peter";

names[3] = "scott";


// Alternative way to initialize an array

let names2: string[] = ["john", "smith", "peter", "scott"];
```

Approach 2: Generic Array Syntax

```
let empNames: Array<string> = ["john", "smith", "peter", "scott"]; // Array of strings

let empIds: Array<number> = [101, 102, 103, 104]; // Array of numbers

let data: Array<string | number> = ["john", "smith", 101, 102]; // Union type (string or number)

let data2: Array<any> = [1, "john", true, null]; // Array allowing multiple data types
```

Accessing Array Elements

- **Access by index** (indexing starts from 0).
- **Printing arrays** using console.log().

```
console.log(names);    // Output: ['john', 'smith', 'peter', 'scott']

console.log(names[1]); // Output: smith

console.log(names[4]); // Output: undefined (index out of bounds)
```

Iterating Over an Array

There are multiple ways to loop through an array:

Using a for loop

```
for (let i = 0; i < empNames.length; i++) {
    console.log(empNames[i]);
}
```

Using for...in loop (indexes)

```
for (let i in empIds) {
    console.log(empIds[i]); // 'i' represents index
}
```

Using for...of loop (values)

```
for (let element of data) {
    console.log(element); // 'element' represents actual array values
}
```

Passing an Array to a Function

Arrays can be passed to functions for processing.

Example: Searching for an Element in an Array

```
function search(ele: number, arr: number[]): boolean {
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] === ele) {
      return true; // Element found
    }
  }
  return false; // Element not found
}
```

```
let arr: number[] = [10, 20, 30, 40, 50];
console.log(search(20, arr));  // Output: true
console.log(search(100, arr)); // Output: false
```

Function Returning an Array

A function can take an array as input and return a modified array.

Example: Capitalizing Words in an Array

```
function capitalizeWords(arr: string[]): string[] {
   let result: string[] = [];
   for (let i = 0; i < arr.length; i++) {
      result[i] = arr[i].toUpperCase();
   }
   return result;
}

let words: string[] = ["hello", "world", "typescript"];
console.log(capitalizeWords(words)); // Output: ["HELLO", "WORLD", "TYPESCRIPT"]
```

Key Takeaways

- Arrays store multiple values and can hold elements of the same or mixed types.
- Two ways to declare arrays: **array literals ([])** and **generic syntax (Array<Type>)**.
- Array indexing starts from 0.
- Elements can be accessed, modified, and iterated using loops (for, for...in, for...of).
- Arrays can be passed to functions for processing.
- Functions can return modified arrays.

Tuple – A Special Type of Array

A tuple is a fixed-length array where each element has a specific type.

It helps in storing multiple fields of different data types together.

```
let person: [string, number] = ["Alice", 25];

console.log(person[0]); // Output: Alice

console.log(person[1]); // Output: 25
```