■ JavaScriptの配列（Array）

基本構文

```
const fruits: string[] = ['apple', 'banana', 'orange'];
console.log(fruits[0]); // apple
```

特徴

- 長さ可変（Javaの `ArrayList` に近い）
- 型はバラバラでもOK（ `[1, "text", true]` のように混在可能）
- 配列の操作に便利なメソッドが多数（後述）

よく使うメソッド

メソッド	説明	例
`push()`	配列の末尾に要素を追加	`arr.push('grape')`
`map()`	各要素に処理を適用して新配列	`arr.map(x => x * 2)`
`filter()`	条件に合う要素を抽出	`arr.filter(x => x > 5)`
`forEach()`	各要素に対して副作用あり処理	`arr.forEach(x => console.log(x))`
`find()`	条件に合う最初の要素を取得	`arr.find(x => x === 'banana')`

■ JavaScriptのオブジェクト（Object）

基本構文

```
const user: { name: string; age: number } = {
  name: 'Taro',
  age: 30,
};

console.log(user.name); // Taro
```

特徴

- JavaでいうMapやクラスのように使える
- プロパティは自由に追加・削除可能
- オブジェクト同士の入れ子も簡単

アクセス方法

```
console.log(user['age']); // 30
(user as any)['gender'] = 'male'; // プロパティの追加
```

ネスト（入れ子）

```
const person: {
  name: string;
  address: {
    city: string;
    zip: string;
  };
} = {
  name: 'Hanako',
  address: {
    city: 'Tokyo',
    zip: '100-0001',
  },
};

console.log(person.address.city); // Tokyo
```

■ Javaとの違いの補足

項目	Java	JavaScript
配列	型固定	型自由
オブジェクト	クラス定義が必要	直接リテラルで作成可能
コレクション操作	for文、Streamなど	map, filter, forEachなど

■ JavaScriptの関数

関数宣言（function文）

```
function greet(name: string): string {
  return `Hello, ${name}`;
}
console.log(greet('Taro')); // Hello, Taro
```

無名関数（関数式）

JavaScriptでは、関数を**変数に代入して扱う**こともできます：

```
const greet = function (name: string): string {
  return `Hello, ${name}`;
};
console.log(greet('Taro')); // Hello, Taro
```

> Javaでいう「関数型インターフェース + ラムダ式」に近い考え方。

アロー関数（ES6以降）

無名関数を**より簡潔に書ける**構文：

```
const greet = (name: string): string => {
  return `Hello, ${name}`;
};
```

さらに短くも書けます（戻り値が1行の場合）：

```
const greet = (name: string): string => `Hello, ${name}`;
```

ポイントまとめ

- 関数は**オブジェクトと同様に変数に代入可能**
- 引数や戻り値の**型定義はない（動的型付け）**
- `const` と組み合わせて関数を定義するのが主流

■ まとめ：関数定義の比較

種類	書き方	特徴
関数宣言	`function greet() {}`	どこでも呼び出せる（巻き上げ）
無名関数	`const greet = function() {}`	柔軟、巻き上げされない
アロー関数	`const greet = () => {}`	より簡潔、`this`の挙動に注意

■ おまけ：配列との組み合わせ（よくある書き方）

```
const numbers: number[] = [1, 2, 3];
const doubled: number[] = numbers.map((num) => num * 2);
console.log(doubled); // [2, 4, 6]
```

■ スプレッド構文（spread syntax）

基本構文（配列）

スプレッド構文（`...`）を使うと、**配列を展開して別の配列にコピーや結合が可能**です：

```
const arr1: number[] = [1, 2];
const arr2: number[] = [...arr1, 3, 4];
console.log(arr2); // [1, 2, 3, 4]
```

基本構文（オブジェクト）

オブジェクトのコピーやマージも可能です：

```
const user = { name: 'Taro', age: 30 };
const newUser = { ...user, gender: 'male' };
console.log(newUser); // { name: 'Taro', age: 30, gender: 'male' }
```

> 後に書いたプロパティが**上書きされる**点に注意：

```
const updatedUser = { ...user, age: 35 };
console.log(updatedUser); // { name: 'Taro', age: 35 }
```

特徴

- 配列・オブジェクトを**浅くコピー（shallow copy）**
- 元の値は**変更されない（イミュータブルな処理）**
- **関数の引数展開**にも使える：

```
const nums: number[] = [1, 2, 3];
console.log(Math.max(...nums)); // 3
```