# Основы TypeScript

# Арифметические операции

- Возведение в степень: `Math.pow(number, exponent)`

- Квадратный корень: `Math.sqrt(number)`

const square = Math.pow(number, 2);

const squareRoot = Math.sqrt(number);

# Ресурсы

- Официальная документация TypeScript: [https://www.typescriptlang.org/docs/](https://www.typescriptlang.org/docs/)

# Типы переменных

- В TypeScript переменные имеют типы, которые определяют, какие значения могут быть присвоены переменной.

- Примеры типов: `number`, `string`, `boolean`, `any`, `unknown`, `never`, `void`.

let num: number = 42;

let str: string = "Hello";

let isTrue: boolean = true;

// Получение типа переменной

console.log(typeof num); // Вывод: "number"

console.log(typeof str); // Вывод: "string"

console.log(typeof isTrue); // Вывод: "boolean"

```

# Функции

- Функции могут возвращать множественные значения через кортежи.

function scaleEtc(

vers: { [key: string]: number[][] },

ver: string,

maxx: number,

maxy: number

): [number, number, number, number, number] {

const vermaxx = Math.max(...vers[ver][0]);

const verminx = Math.min(...vers[ver][0]);

const vermaxy = Math.max(...vers[ver][1]);

const verminy = Math.min(...vers[ver][1]);

const vercenx = (vermaxx + verminx) / 2;

const verceny = (vermaxy + verminy) / 2;

const verdiffx = vermaxx - verminx;

const verdiffy = vermaxy - verminy;

const scalex = maxx / verdiffx;

const scaley = maxy / verdiffy;

const scale = Math.min(scalex, scaley);

return [verminx, verminy, scale, vercenx, verceny];

}

const [verminx, verminy, scale, vercenx, verceny] = scaleEtc(vers, ver, maxx, maxy);

```

# Классы

- Классы позволяют создавать объекты с определенными свойствами и методами.

- Статические методы и поля принадлежат классу, а не экземпляру класса.

class Parent {

method() {

console.log("Parent method");

}

}

class Child extends Parent {

override method() {

super.method();

console.log("Child method additional logic");

}

}

const child = new Child();

child.method(); // Вывод: "Parent method", затем "Child method additional logic"

```

# Структуры данных

## Массивы

- Статический массив: массив с фиксированным размером.

- Динамический массив: массив, размер которого можно изменять.

let staticArray: number[] = [1, 2, 3];

let dynamicArray: number[] = [];

dynamicArray.push(1);

dynamicArray.push(2);

dynamicArray.push(3);

```

## Многомерные массивы

let multiDimensionalArray: number[][] = [

[1, 2, 3],

[4, 5, 6]

];

## Зубчатый массив

let jaggedArray: number[][][] = new Array(2);

for (let i = 0; i < 2; i++) {

jaggedArray[i] = new Array(3);

for (let j = 0; j < 3; j++) {

jaggedArray[i][j] = new Array(3);

}

}

## Списки

let list: number[] = [];

list.push(1);

list.push(2);

list.push(3);

list.pop(); // Удаляет последний элемент

list.shift(); // Удаляет первый элемент

## Словари

let dictionary: { [key: string]: number } = {};

dictionary["key1"] = 1;

dictionary["key2"] = 2;

console.log(dictionary["key1"]); // Вывод: 1

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Array<T>

Array<T> — это тип массива в TypeScript, где T указывает на тип элементов, которые будут храниться в массиве.

Массивы в TypeScript могут хранить элементы одного типа, что обеспечивает типобезопасность.

#### Создание и инициализация массива

*// Массив чисел*

let numbers: Array<number> = [1, 2, 3, 4, 5];

*// Массив строк*

let strings: Array<string> = ["apple", "banana", "cherry"];

*// Массив объектов*

let objects: Array<{ name: string; age: number }> = [

{ name: "Alice", age: 25 },

{ name: "Bob", age: 30 },

{ name: "Charlie", age: 35 }

];

#### Доступ к элементам массива

console.log(numbers[0]); *// Вывод: 1*

console.log(strings[1]); *// Вывод: "banana"*

console.log(objects[2].name); *// Вывод: "Charlie"*

#### Добавление нового элемента в массив

numbers.push(6); *// Добавляет число 6 в конец массива*

strings.push("date"); *// Добавляет строку "date" в конец массива*

objects.push({ name: "Diana", age: 40 }); *// Добавляет объект в конец массива*

console.log(numbers); *// Вывод: [1, 2, 3, 4, 5, 6]*

console.log(strings); *// Вывод: ["apple", "banana", "cherry", "date"]*

console.log(objects); *// Вывод: [{ name: "Alice", age: 25 }, { name: "Bob", age: 30 }, { name: "Charlie", age: 35 }, { name: "Diana", age: 40 }]*

#### Удаление элементов из массива

numbers.pop(); *// Удаляет последний элемент (6)*

strings.shift(); *// Удаляет первый элемент ("apple")*

objects.splice(1, 1); *// Удаляет элемент с индексом 1 ({ name: "Bob", age: 30 })*

console.log(numbers); *// Вывод: [1, 2, 3, 4, 5]*

console.log(strings); *// Вывод: ["banana", "cherry", "date"]*

console.log(objects); *// Вывод: [{ name: "Alice", age: 25 }, { name: "Charlie", age: 35 }, { name: "Diana", age: 40 }]*

#### Итерация по массиву

numbers.forEach((num) => {

console.log(num);

});

*// Вывод: 1, 2, 3, 4, 5*

strings.forEach((str) => {

console.log(str);

});

*// Вывод: "banana", "cherry", "date"*

objects.forEach((obj) => {

console.log(`Name: ${obj.name}, Age: ${obj.age}`);

});

*// Вывод:*

*// Name: Alice, Age: 25*

*// Name: Charlie, Age: 35*

*// Name: Diana, Age: 40*

## Record<K, T>

Record<K, T> — это утилитный тип в TypeScript, который используется для создания объекта с определенными ключами и значениями.

K — тип ключей объекта (обычно строка или символ).

T — тип значений объекта.

### Создание и инициализация объекта

*// Объект со строковыми ключами и строковыми значениями*

let obj1: Record<string, string> = {

key1: "value1",

key2: "value2",

key3: "value3"

};

*// Объект со строковыми ключами и числовыми значениями*

let obj2: Record<string, number> = {

key1: 1,

key2: 2,

key3: 3

};

*// Объект со строковыми ключами и объектными значениями*

let obj3: Record<string, { name: string; age: number }> = {

key1: { name: "Alice", age: 25 },

key2: { name: "Bob", age: 30 },

key3: { name: "Charlie", age: 35 }

};

### Доступ к элементам объекта

console.log(obj1.key1); *// Вывод: "value1"*

console.log(obj2.key2); *// Вывод: 2*

console.log(obj3.key3.name); *// Вывод: "Charlie"*

### Добавление новых ключей и значений

obj1.key4 = "value4"; *// Добавляет новый ключ "key4" со значением "value4"*

obj2.key4 = 4; *// Добавляет новый ключ "key4" со значением 4*

obj3.key4 = { name: "Diana", age: 40 }; *// Добавляет новый ключ "key4" со значением { name: "Diana", age: 40 }*

console.log(obj1); *// Вывод: { key1: "value1", key2: "value2", key3: "value3", key4: "value4" }*

console.log(obj2); *// Вывод: { key1: 1, key2: 2, key3: 3, key4: 4 }*

console.log(obj3); *// Вывод: { key1: { name: "Alice", age: 25 }, key2: { name: "Bob", age: 30 }, key3: { name: "Charlie", age: 35 }, key4: { name: "Diana", age: 40 } }*

### Итерация по объекту

for (const key in obj1) {

console.log(`Key: ${key}, Value: ${obj1[key]}`);

}

*// Вывод:*

*// Key: key1, Value: value1*

*// Key: key2, Value: value2*

*// Key: key3, Value: value3*

*// Key: key4, Value: value4*

for (const key in obj3) {

console.log(`Key: ${key}, Value: ${obj3[key].name}, Age: ${obj3[key].age}`);

}

*// Вывод:*

*// Key: key1, Value: Alice, Age: 25*

*// Key: key2, Value: Bob, Age: 30*

*// Key: key3, Value: Charlie, Age: 35*

*// Key: key4, Value: Diana, Age: 40*

### Динамические ключи

Если вам нужно использовать динамические ключи, Record<string, T> подходит идеально.

let obj4: Record<string, any> = {

key1: "value1",

key2: 2,

key3: true,

key4: { nestedKey: "nestedValue" }

};

# Многопоточность

- TypeScript работает в однопоточном режиме, но можно использовать Web Workers для асинхронных операций.

const worker = new Worker("worker.js");

worker.onmessage = function (event) {

console.log("Message from worker:", event.data);

};

worker.postMessage("Hello from main thread");