ЗАЩИТНИК ТИПА (TYPE GUARD)

Когда мы изучали тему **сужения типов**, то мы создавали различные условия для их определения. Правила, которые позволяют выводу типов определить **суженый диапазон типов** для значения называются **защитниками типа, type guards**

```
function printMsg(msg: string[] | number | boolean): void {
    if (Array.isArray(msg)) {  // <= type guard
        msg.forEach((m) => console.log(m));
    } else if (typeof msg === "number") { // <= type guard
        console.log(msg);
    } else {
        console.log(msg);
    }
}
```

Для соблюдения принципа **DRY** (не повторяем код) мы такие правила можем вынести в отдельную функцию. В TS можно использовать дополнительный синтаксис для таких функций и создавать **пользовательские защитники типа**, которые возвращают предикат *(лат. praedicatum, логическое значение: true или false)*

```
function isNumber(n: unknown): n is number {
    return typeof n === "number";
}
```

```
function isNumber(n: string[] | number | boolean): n is number {
    return typeof n === "number";
}
```

Функция inNumber вернет true только если аргумент будет числом. **Оператор is** позволяет сказать, что будет возвращено **логическое** значение, где проверяется , что n это число. Дальше используем её в условии:

```
if (Array.isArray(msg)) {
    msg.forEach((m) => console.log(m));
} else if (isNumber(msg)) {
    console.log(msg);
} else {
```

ЗАЩИТНИК ТИПА (TYPE GUARD)

Более продвинутые защитники определяют, к какому объекту относится эта сущность. Например, по интерфейсу:

```ts
interface Car {
    engine: string;
    wheels: number;
}

interface Ship {
    engine: string;
    sail: string;
}

function isCar(car: Car | Ship): car is Car {
    return "wheels" in car;
}

function isShip(ship: Car | Ship): ship is Ship {
    return "sail" in ship;
}

function repairVehicle(vehicle: Car | Ship) {
    if (isCar(vehicle)) {
        vehicle.wheels;
    } else if (isShip(vehicle)) {
        vehicle.sail;
    } else {
        vehicle;
    }
}
```

Тут мы узнаем, есть ли такое свойство в интерфейсе и возвращаем true, если оно есть. Если нет - false. За счет такой проверки, внутри функции мы получаем корректные подсказки о том, что это за объект.

Но вы можете создавать **любые условия** внутри пользовательских защитников для ваших целей. Более продвинутый вариант функции выше, позволяет работать со вложенными структурами:

```ts
function isCar(car: Car | Ship): car is Car {
    return (car as Car).wheels !== undefined;
}
```

```ts
function isCar(car: Car | Ship): car is Car {
    return (car as Car).wheels.number !== undefined;
}
```