УТВЕРЖДЕНИЕ ТИПОВ

В реальном коде случаются ситуации, когда полученное откуда-то значение не соответствует тому типу, который мы ожидаем. Для того, чтобы попросить TS **пересмотреть** свое отношение к определенному типу, используется механизм **утверждения типа**

```ts
const fetchData = (url: string, method: "GET" | "POST"): void => {
    console.log(method);
};

const reqOptions = {
    url: "https://someurl.com",
    method: "GET",
};

fetchData("qqq", "GET");
fetchData(reqOptions.url, reqOptions.method); // Error
```

Свойство **reqOptions.method** имеет тип string, а значит не подходит под аннотацию аргумента с литералами. Если мы **точно** знаем, что значение свойства подходит, то мы можем утвердить это значение оператором **as**:

```ts
const fetchData = (url: string, method: "GET" | "POST"): void => {
    console.log(method);
};

const reqOptions = {
    url: "https://someurl.com",
    method: "GET",
};

fetchData("qqq", "GET");
fetchData(reqOptions.url, reqOptions.method as "GET");
```

Что **важно** помнить:

👉 В таких операциях **вы берете риск на себя.** TS позволяет вам уточнить строку в её литерал (конкретное значение). **Но при изменении значения свойства** в объекте TS вам не укажет на ошибку

👉 TS **не позволит** вам создавать нелогичные конструкции. Вы просите пересмотреть отношение, но **не указываете воспринимать** так-то. Доступно лишь утверждение **более специализированных типов**: строка - её литерал, число - его литерал и тп. В обратную сторону операция не имеет смысла

```ts
fetchData(reqOptions.url, reqOptions.method as 5); // Error
```

👉 Необходимо отличать **утверждение типов** от **преобразования типов.** От конструкции с as после компиляции **ничего** не останется. Это просто просьба того, каким типом считать эту сущность при разработке. Преобразование - это превращение числа в строку и тп.

АЛЬТЕРНАТИВНЫЕ ВАРИАНТЫ

Эту ситуацию можно решить еще двумя другими способами, так же утверждением типов. **Первый** - утвердить значение еще на этапе объекта:

```
const reqOptions = {
    url: "https://someurl.com",
    method: "GET" as "GET",
};
```

Это удобно, так как и утверждение и значение находятся в одном месте. **Но**, если свойство используется в разных места по разному, это может навредить. Иногда приходится утверждать тип в момент использования. **Второй** - превратить **весь объект в литерал типа:**

```
const reqOptions = {
    url: "https://someurl.com",
    method: "GET",
} as const;
```

Напоминание! Если переменная с примитивом создается без аннотации и через const, то вывод типов присваивает ей литеральный тип. С ообъектами так не работает, но можно сделать через **as const**

Существует **альтернативный синтаксис** утверждения типов, через угловые скобки. Он **не так удобен** и **не работает** в некоторых технологиях, где идет конфликт по скобкам (например React):

```
fetchData(reqOptions.url, <"GET">reqOptions.method);
```

Часто утверждение типов можно встретить при работе с DOM-деревом, когда мы хотим уточнить, с каким элементом мы работаем:

```
const box = document.querySelector(".box") as HTMLElement;
const input = document.querySelector("input") as HTMLInputElement;
// Альтернативный:
const input = <HTMLInputElement>document.querySelector("input");
```

Мы “уточняем” более специализированный интерфейс, ведь **HTMLElement это частный случай Element** (будет рассмотрено в следующих уроках)

Важно помнить, что элемент может быть не найден на странице, так что использование различных конструкций по предотвращению ошибок обязательно (try/catch, if и тп.)