- **Модуль:** Typescript. Generics and type manipulations
- **Урок:** Generics (Обобщения), что это и зачем нужно

TypeScript Handbook
from
Иван Петриченко
udemy

ОБОБЩЕНИЯ

В программировании постоянно создаются конструкции, которые можно **переиспользовать** в разных ситуациях с разными данными. В стандартном JS мы просто передавали в функцию аргумент и в теле уже с ним работали. В TS необходимо **указать тип аргумента**

```
function processingData(data) {
    //...

    return data;
}
```

← Функция, принимающие данные в виде строки, числа, логического значения или массива строк

Плохой вариант её типизации

↓

```
function processingData(data: string | number | boolean | string[]): string | number | boolean | string[] {}
```

Для таких ситуаций в TS существует **механизм обобщений** (*generics, дженерики*) Он позволяет поставить **“заглушку”**, которая будет заменена при использовании:

```
function processingData<T>(data: T): T {
    //...

    return data;
}

let res1 = processingData(1); // type number
let res2 = processingData("1"); // type string
```

T - это и есть такая заглушка, которая записывается в угловых скобках и дальше используется для типизации аргумента и возвращаемого значения. При вызове функции мы можем подставить **любой тип** и TS на выходе будет знать, что помещается в результат
Существует **альтернативный синтаксис использования** дженерика, при вызове мы четко говорим что нужно передать в аргумент:

```
const res3 = processingData<number>(10);
```

- **Модуль:** Typescript. Generics and type manipulations
- **Урок:** Generics (Обобщения), что это и зачем нужно

TypeScript Handbook
from
Иван Петриченко
udemy

ПРАВИЛА ИСПОЛЬЗОВАНИЯ ОБОБЩЕНИЙ

👉 Обобщения можно создавать **для типов, интерфейсов, функций, методов и классов**
Для **перечислений** (enum) этого делать нельзя. Про каждый из вариантов мы поговорим в других уроках. Пример с интерфейсом:

```ts
interface Print<Type> {
    design: Type;
}

const somePrint: Print<string> = {
    design: "ten",
};
const someOtherPrint: Print<number> = {
    design: 10,
};
```

👉 Для дженериков можно использовать **любые названия** и обозначения.
Все зависит от сложности:
В простых случаях, обычно, используются буквы **T, U, V, S**. Если там будет property – то **P**, если ключ/значение – то **K/V**
Идентификаторы всегда прописываются с **большой буквы**
В сложных случаях, обычно, эту заглушку описывают более подробно:
ReferralSystem<UserID, UserReferrals>
(количество неограничено, прописываются через запятую)

👉 Существуют **встроенные в TS дженерики** для самых разных целей. Мы касались этой темы при изучении readonly и вот пример создания массива через них:

```ts
Array<T>
const arrOfStrings: Array<string> = ['Hello', 'World!'];
```

На самом деле, когда мы прописываем string[ ], то это удобное сокращение от Array<string>, которое срабатывает внутри TS