# СОЗДАНИЕ TYPE

Как и было указано, можно создавать type, который содержит описание объекта, а не только отдельные литералы. Это позволит нам указывать объектам, что они должны быть одной формы (одного формата, object shape):

```
type Config = { protocol: "http" | "https"; port: 3000 | 3001 };

const serverConfig: Config = {
    protocol: "https",
    port: 3001,
};
```

Теперь type с именем Config можно использовать для аннотирования других объектов. Если они не будут соответствовать этой форме - будет ошибка. В свойствах объекта внутри type может быть что угодно (литералы, типы, объекты и тп.)

В отдельный type можно выносить и описание функции:

```
type StartFunction = (protocol: "http" | "https", port: 3000 | 3001) => string;

const startServer: StartFunction = (
    protocol: "http" | "https",
    port: 3000 | 3001
): "Server started" => {
    console.log(`Server started on ${protocol}://server:${port}`);

    return "Server started";
};
```

# TYPE INTERSECTION

Нам часто приходится комбинировать типы для удобного и быстрого создания нужных нам. Иногда мы не хотим дублировать код (принцип DRY), иногда типы приходят нам из сторонней библиотеки или файла. В этих и других случаях нам понадобится оператор пересечения (&)

Добавим в пример новый тип с ролью, с помощью которого будет создан новый тип конфигурации:

```ts
type Config = { protocol: "http" | "https"; port: 3000 | 3001 };

type Role = {
    role: string;
};

type ConfigWithRole = Config & Role;

const serverConfig: ConfigWithRole = {
    protocol: "https",
    port: 3001,
    role: "admin",
};

const backupConfig: Config = {
    protocol: "http",
    port: 3000
};
```

Благодаря оператору пересечения (intersection, &) мы скомбинировали два типа и получили тип ConfigWithRole. Он содержит все свойства из объединенных типов. Теперь все три type можно использовать в коде