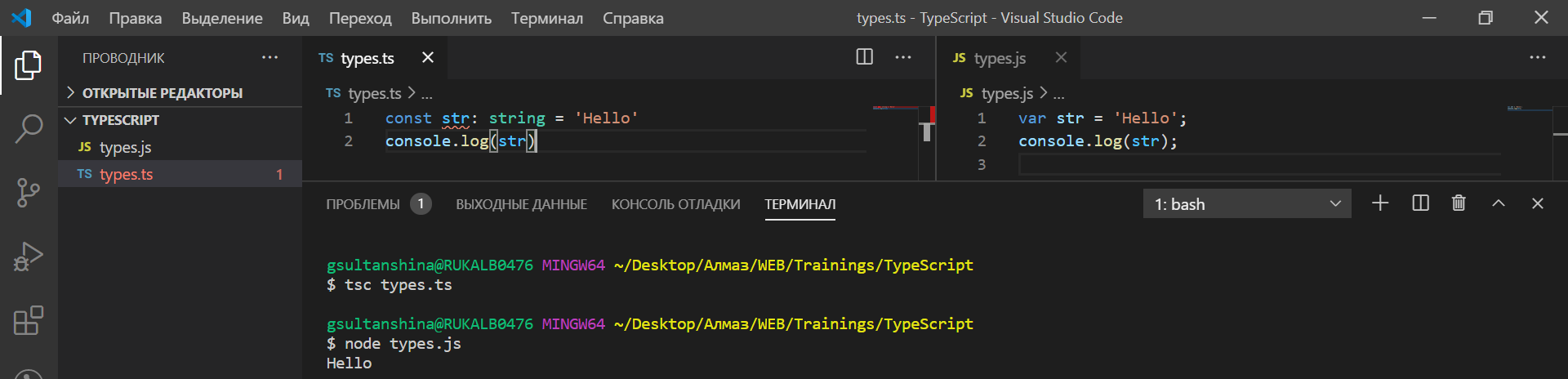
**14.08.2020  
4 часа  
Отчет: TypeScript: Get Started, Handbook by official documentation**

**Что такое TypeScript?**  
TypeScript – это статически-типизируемый JS, используемый для создания сложных приложений  
TypeScript – JavaScript, который масштабируется

**Для чего нужен TypeScript?**  
Изначально JavaScript создавался как скриптовый язык для WEB-страниц, но со временем он начал приобретать популярность и начал использоваться практически везде: при создании мобильных приложений, desktop приложений, игр и тд. JavaScript имеет динамическую типизацию, что является не совсем правильным при создании крупных и сложных проектов. Поэтому Microsoft разработали статический JavaScript и назвали его TypeScript. То есть TypeScript – статический язык программирования, основанный на JavaScript

**Плюсы TypeScript:**  
- отлавливание ошибок на этапе разработки  
- возможность писать сложные приложения с комфортом  
- простой рефакторинг кода  
- удобство при работе в команде  
- богатая документация

**Hello World на TypeScript**Скачаем глобально TypeScript на компьютер с помощью команды: npm install –g typescript  
Создадим файл с расширением .ts  
С помощью команды tsc получим файл с расширением .js  
Запустим js-файл и получим в консоли ожидаемый результат Hello



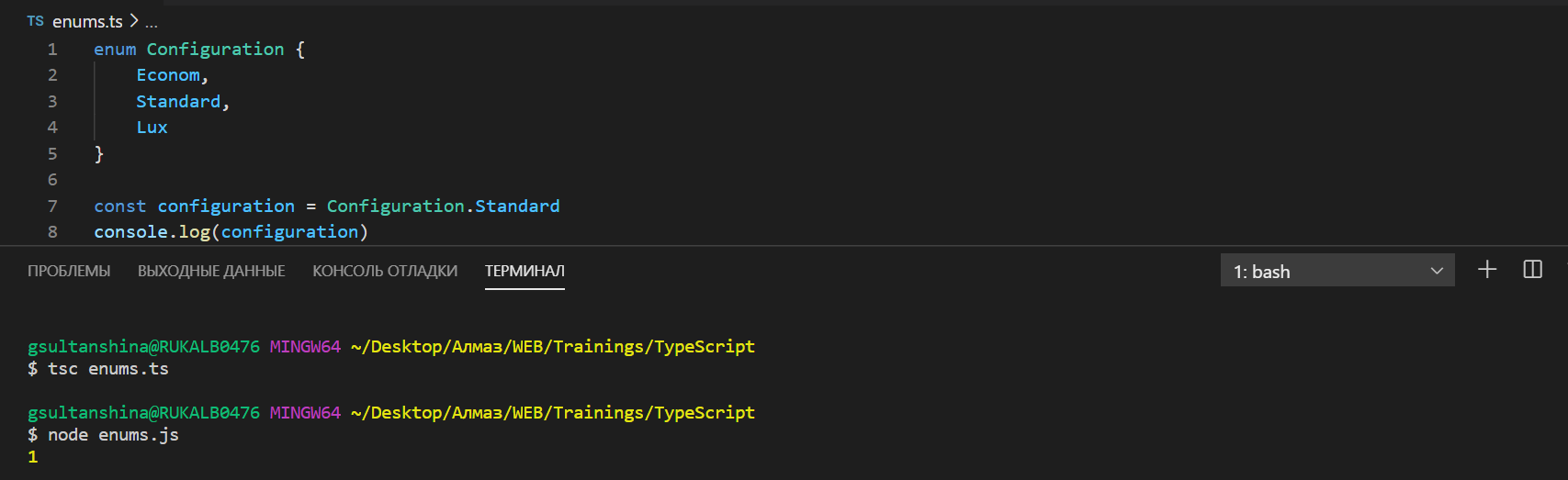
**Интерфейсы**  
Интерфейсы – это тип, который в основном служит для объектов и классов, где мы указываем какие поля, функции, элементы должны присутствовать у этих объектов

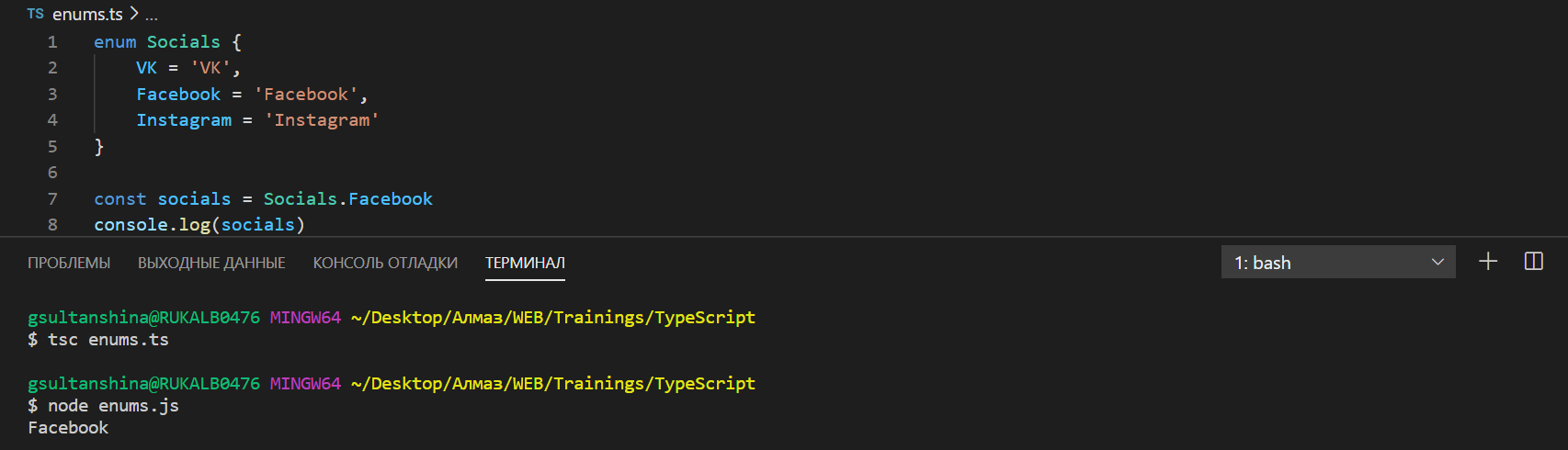
Интерфейсы создаются с помощью ключевого слова interface

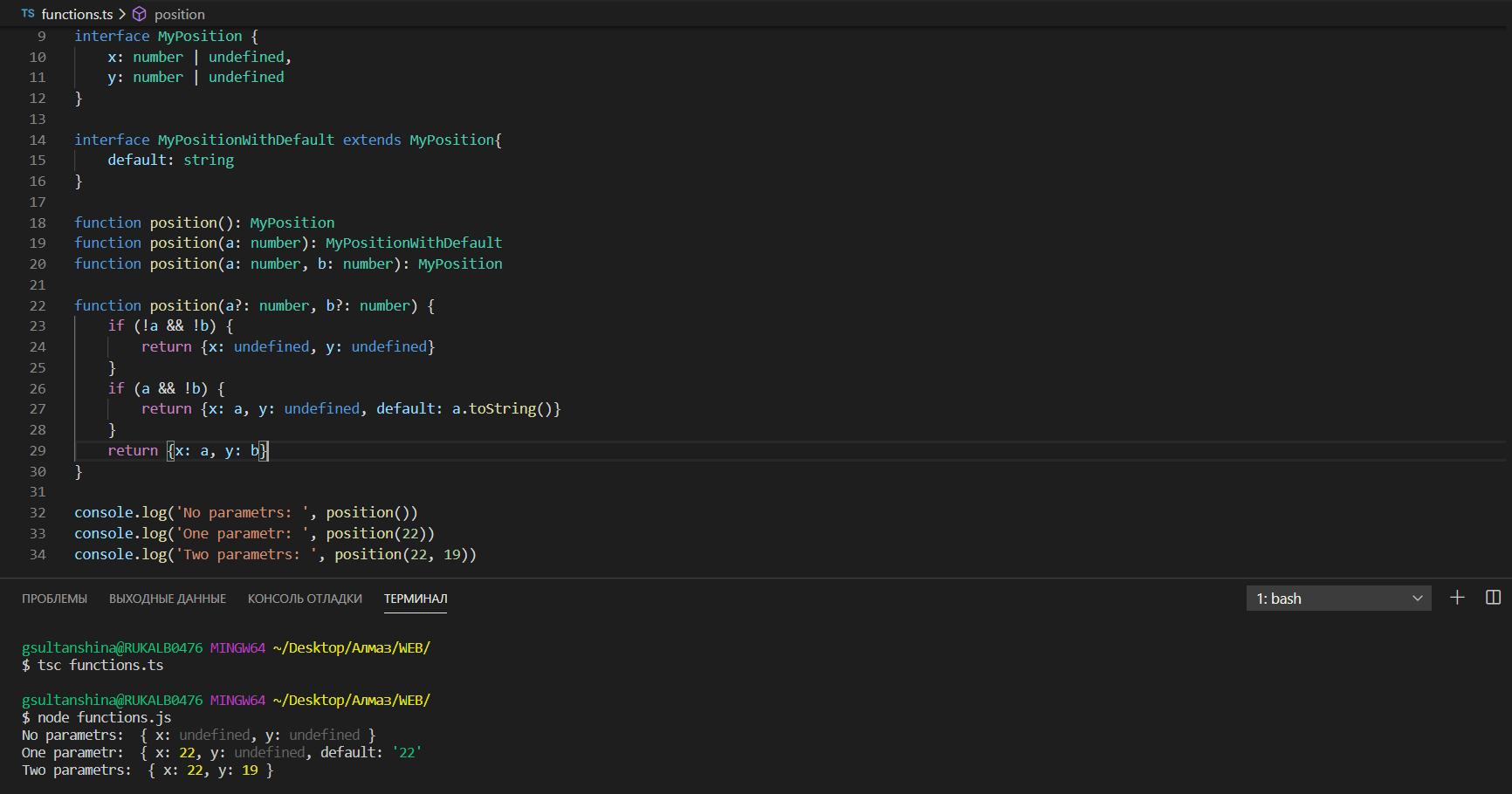
Также их можно наследовать: 

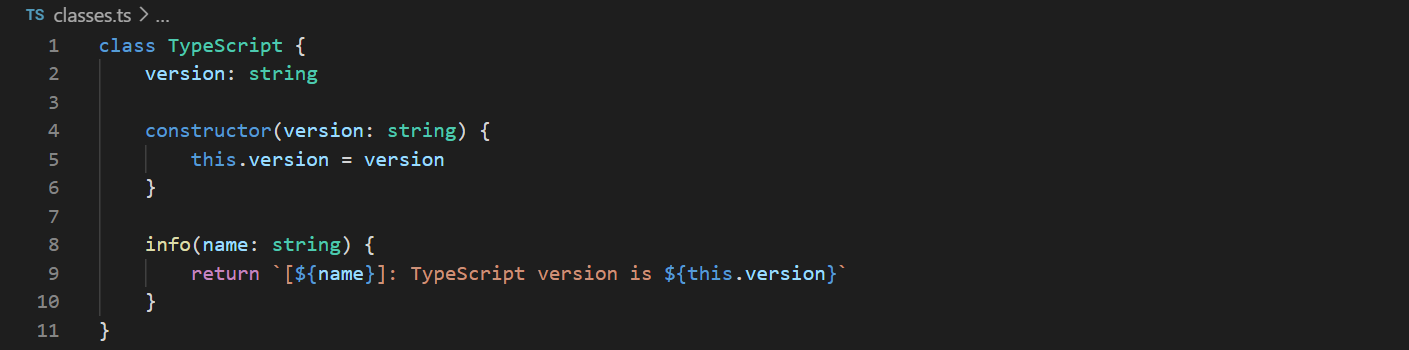
Если нам нужно указывать много свойств, то можно воспользоваться таким приемом: 

**Enums**  
Это вспомогательная сущность, которая позволяет структурировать код, если присутствуют однотипные элементы

У него есть два типа записи: 



**Функции**  


**Классы**Здесь ничего необычного, все то же самое, но есть понимание типа переменных

Создадим еще один класс

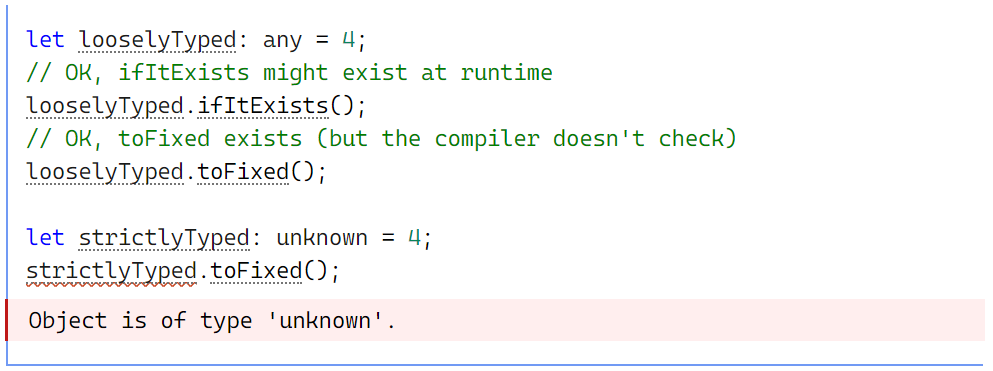
Теперь перезапишем его по-другому

**Модификаторы: protected, public, private**В примере ниже мы создали 3 модификатора класса:  
-public – модификатор по умолчанию, можно менять  
-protected – могут быть доступны в подклассах  
-private – не может быть доступен вне этого класса  

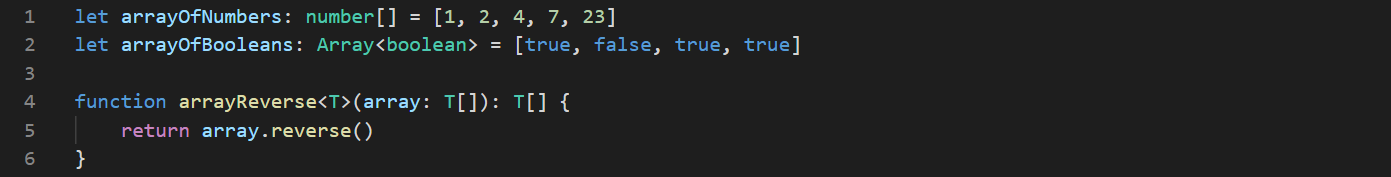

**Абстрактные классы**  
Это базовые классы от которых можно наследоваться   
Абстрактные классы объявляются с помощью ключевого слова abstract

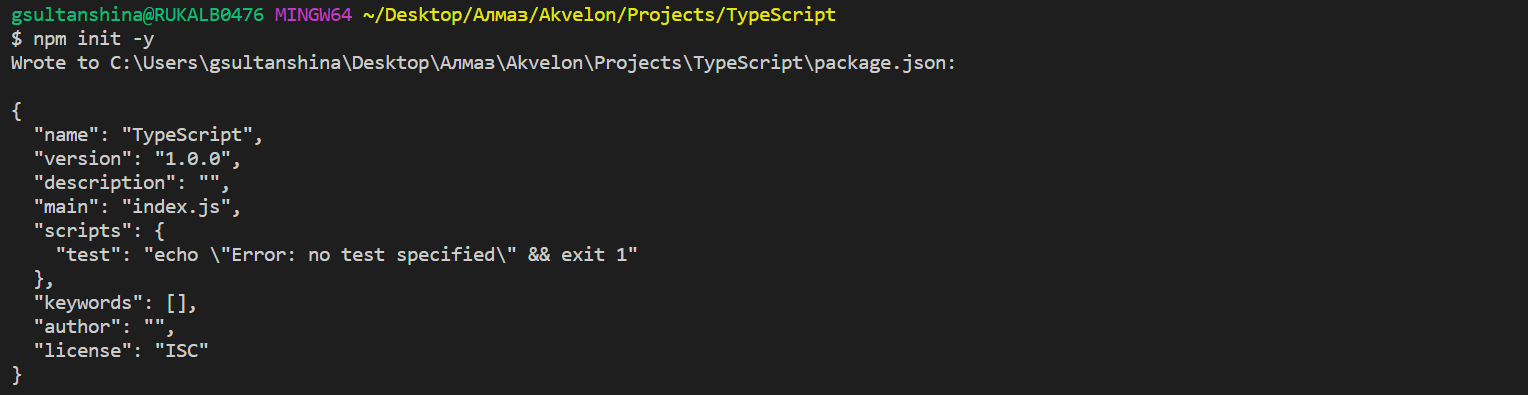
**Guards**  
Это вспомогательные конструкции в TypeScript, которые позволяют работать с типами 

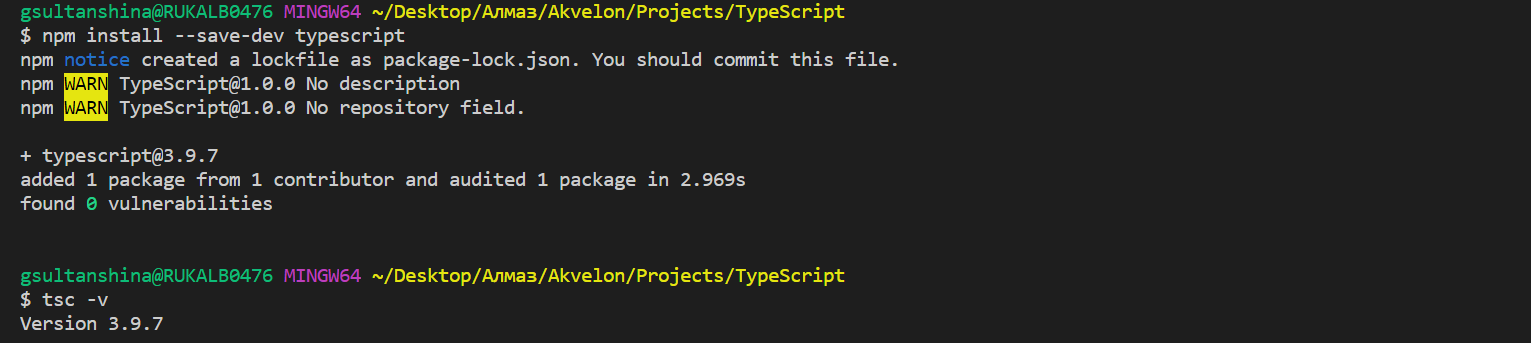
**17.08.2020  
4 часа  
Отчет: TypeScript – Generic functions**

**TypeScript any vs. unknown**В примере ниже мы видим, что переменная типа any «доверяет» нам и пропускает даже несуществующий метод ifItExists(). А переменная типа unknown не «доверяет» и не пропускает даже существующие методы  


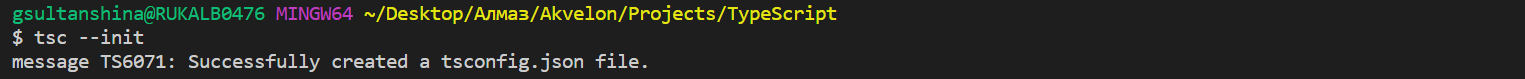
Несмотря на то, что any – удобный тип, пользоваться им рекомендуется крайне редко. Так как TypeScript нужен для «строгой» типизации

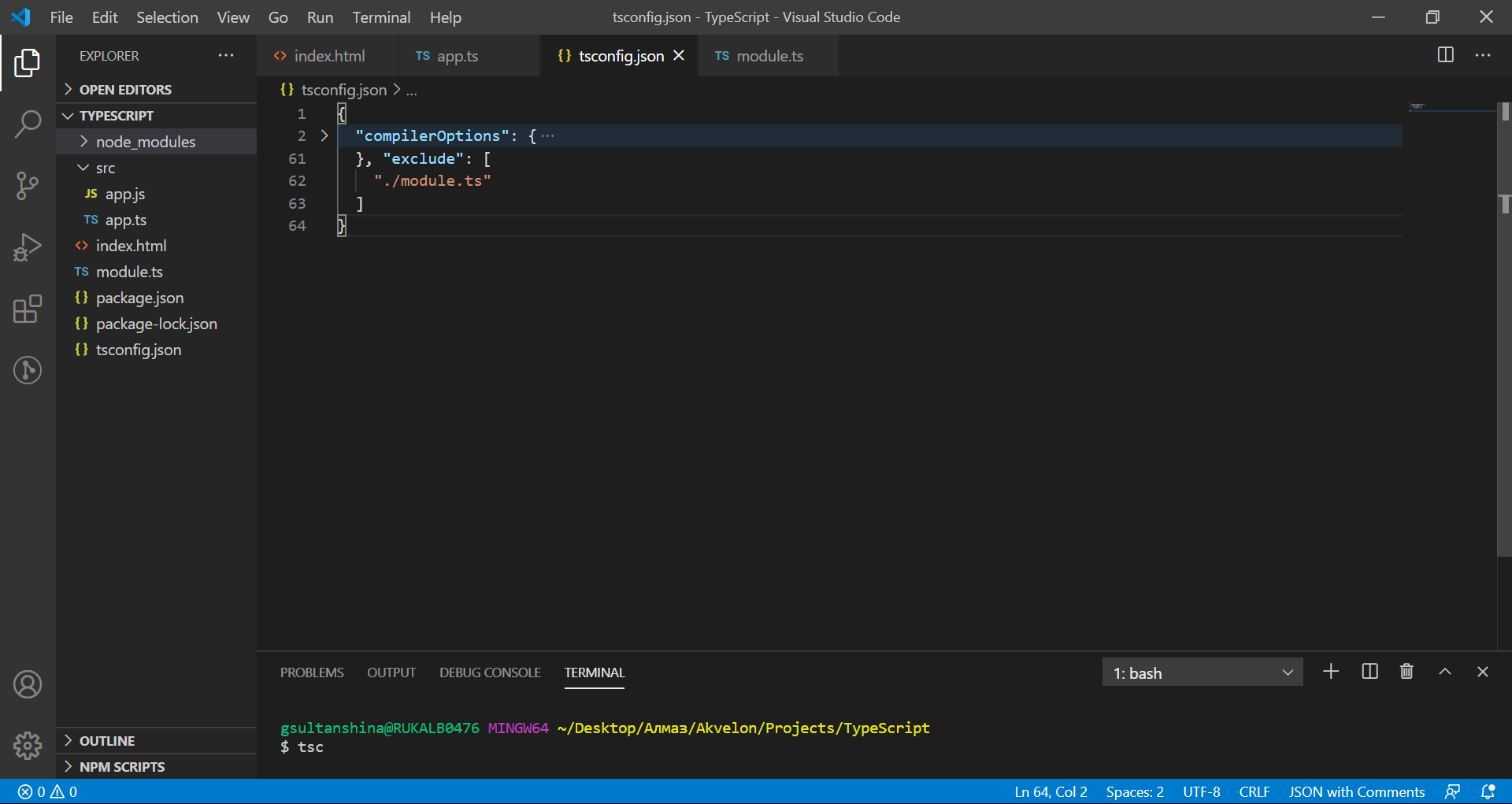
**Generic**В примере ниже мы создали два массива, однако для того, чтобы сделать функцию Reverse на массив мы воспользовались функцией arrayReverse, в которой реализовали Generic Type

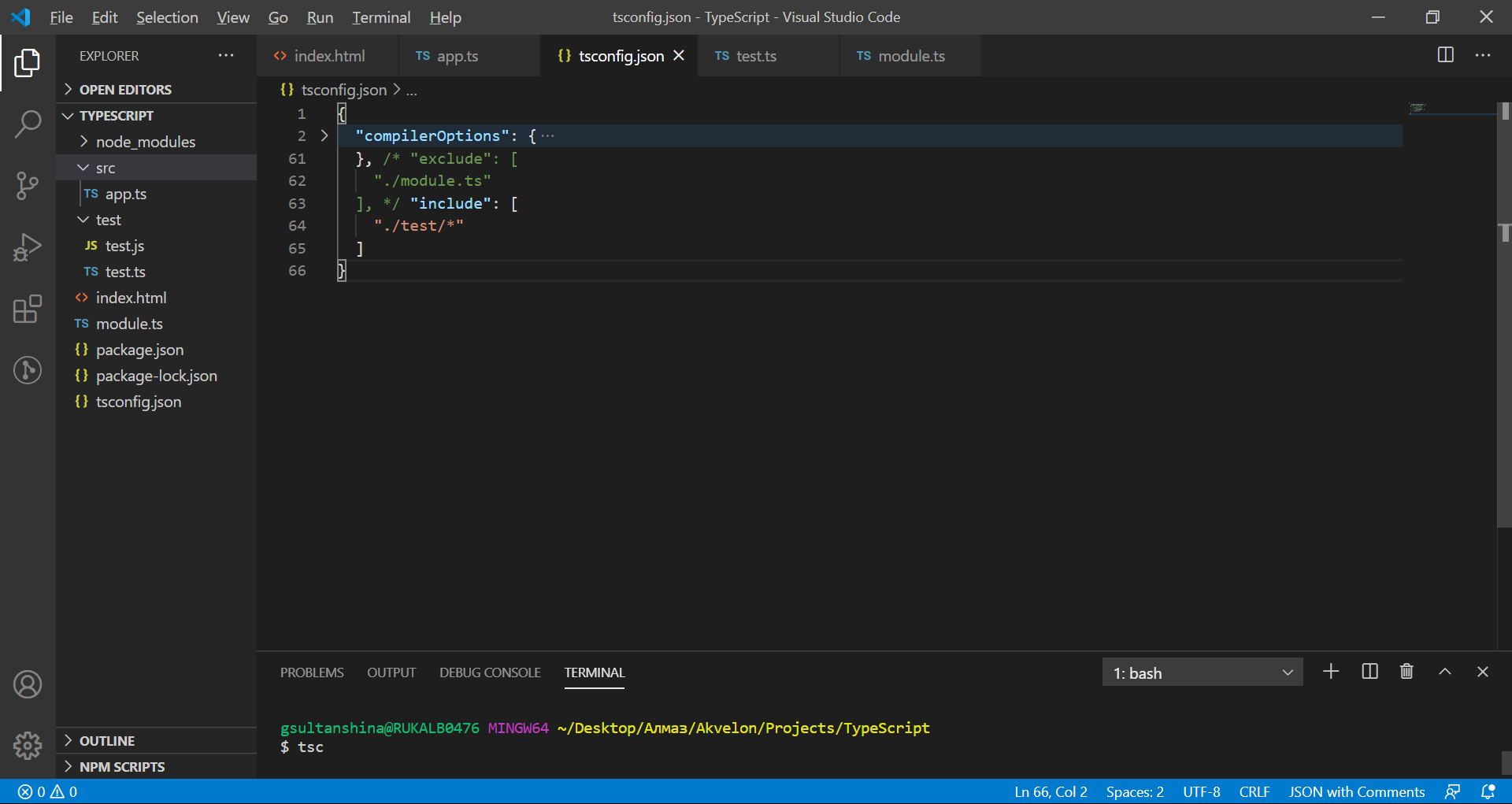
**Practical advanced TypeScript**Проинициализируем проект с помощью команды ниже, получим файл package.json

Теперь установим TypeScript и посмотрим на его версию 

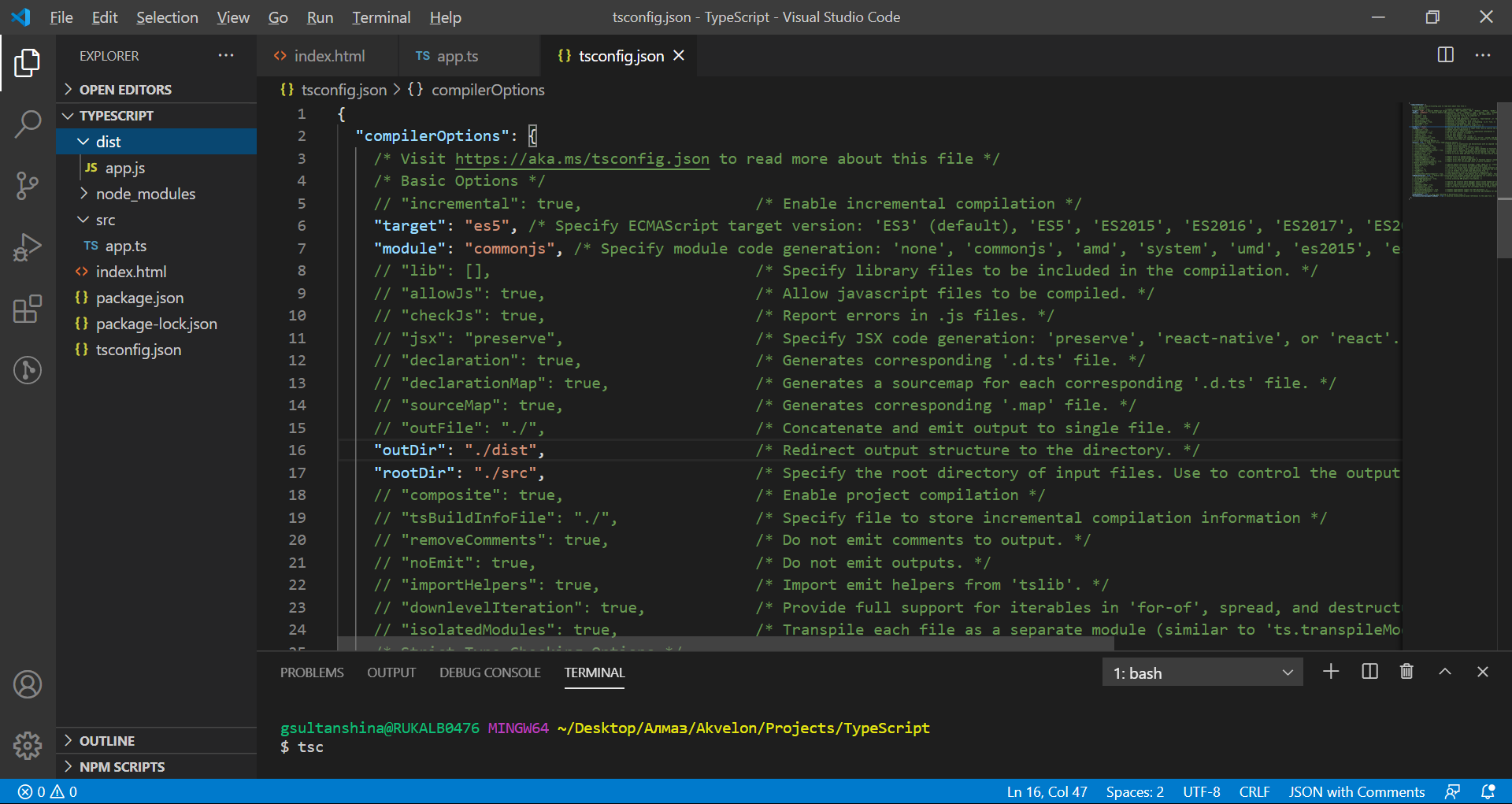
Создадим файл app.ts и скомпилируем его с помощью команды ниже, получим файл app.js 

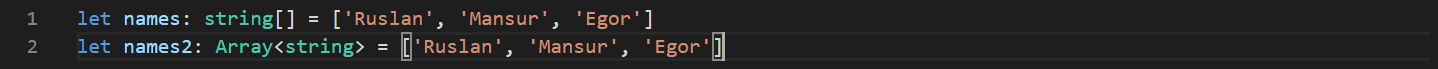
Создадим файл tsconfig.json с помощью команды ниже  


Теперь в tsconfing.json создадим ключ exclude, который отвечает за исключения генерации файлов из ts в js. Внесем туда тот файл или папку, которую мы не хотим отдавать на генерацию. Замечаем, что файл app.ts создал js файл, а файл module.ts – не создал  


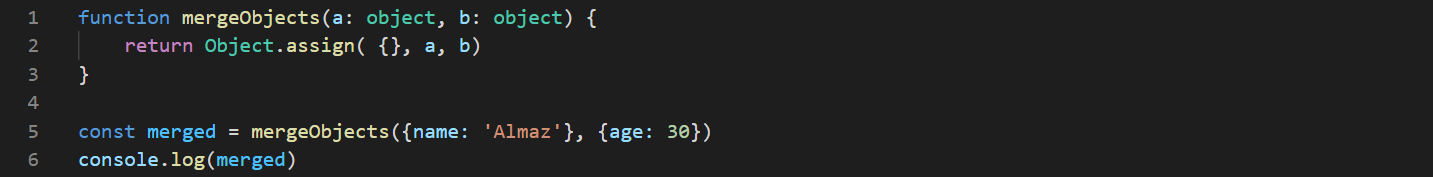
Теперь создадим ключ со значением include, который отвечает за генерацию тех файлов и папок, которые мы перечеслим в этом массиве  


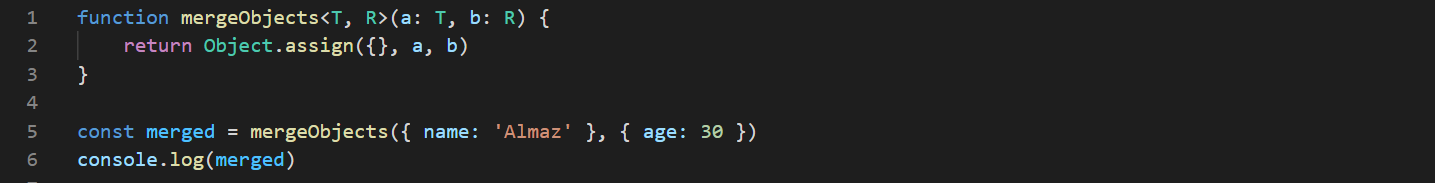
Сделаем небольшие изменения в файле config.json, а именно установим параметры outDir в ./dist, а rootDir в ./src

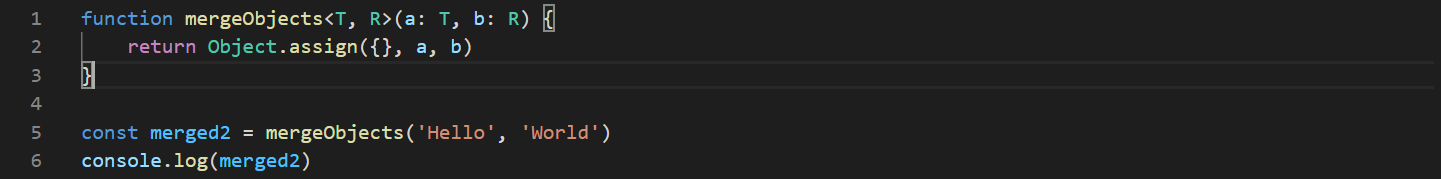


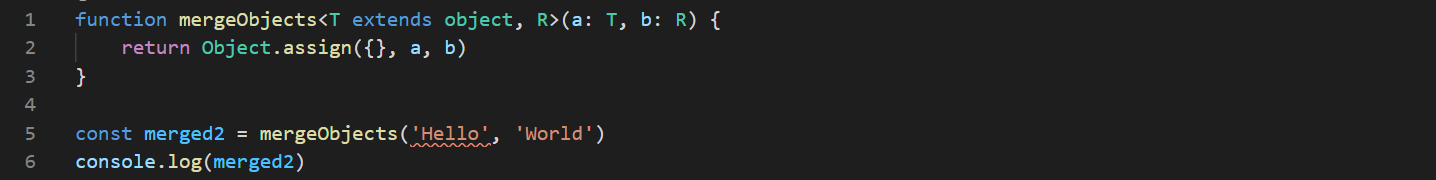
**Generic**  
В примере ниже мы имеем два одинаковых массива, однако names2 имеет тип Generic 

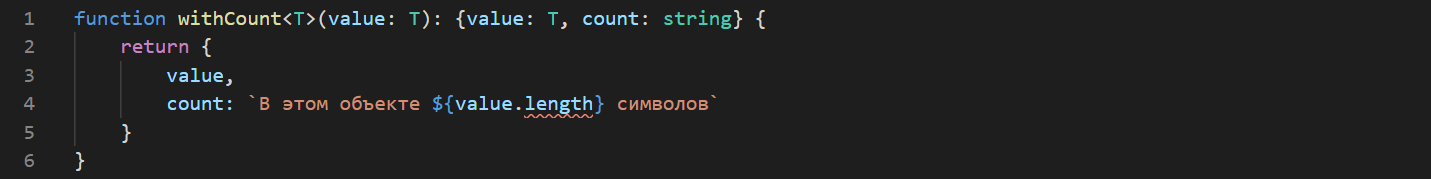
Также в этом примере имеется Generic тип для промиса

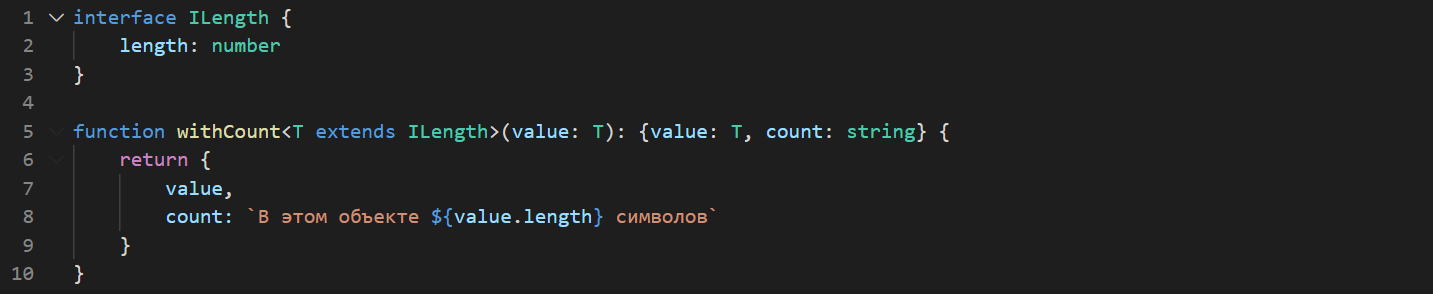
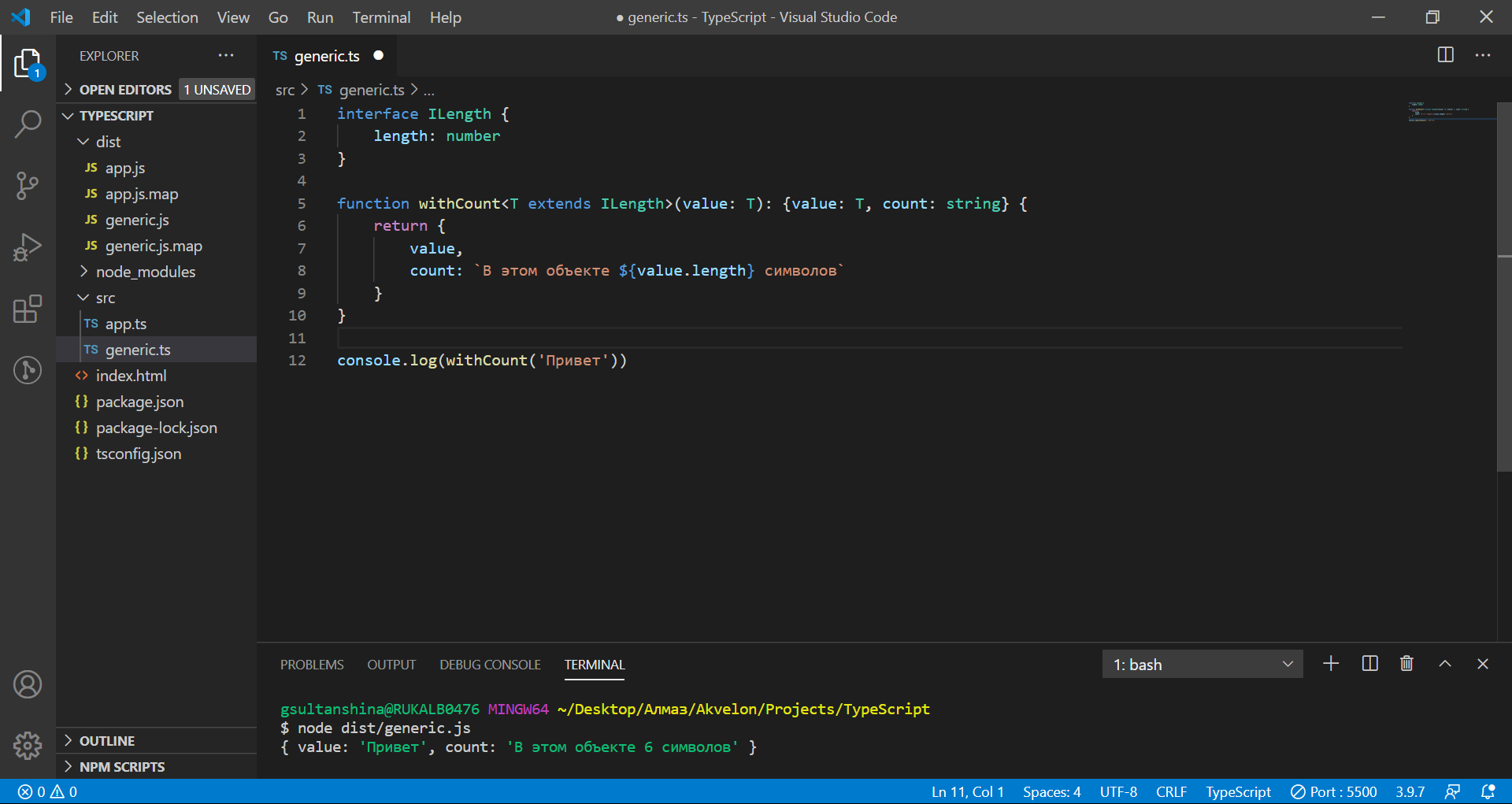
Рассмотрим функцию ниже 

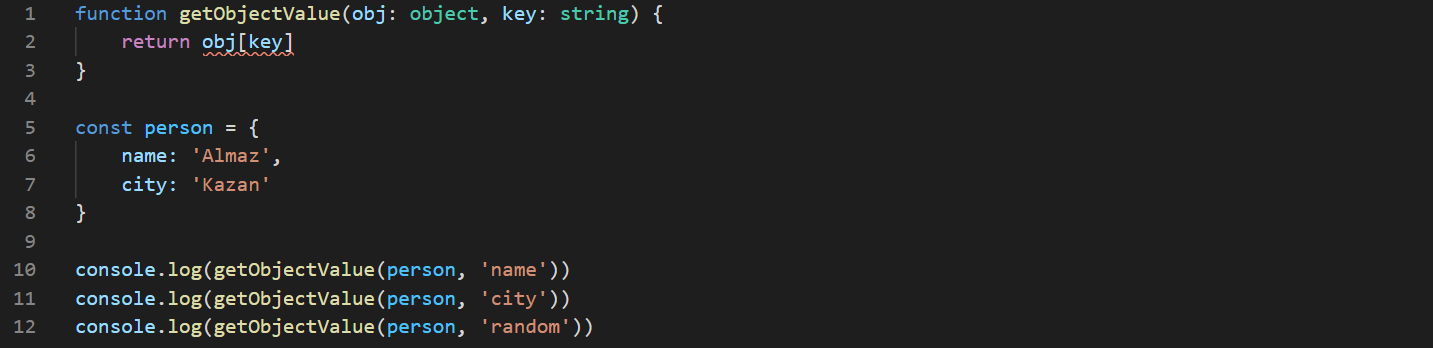
Превратим ее в Generic 

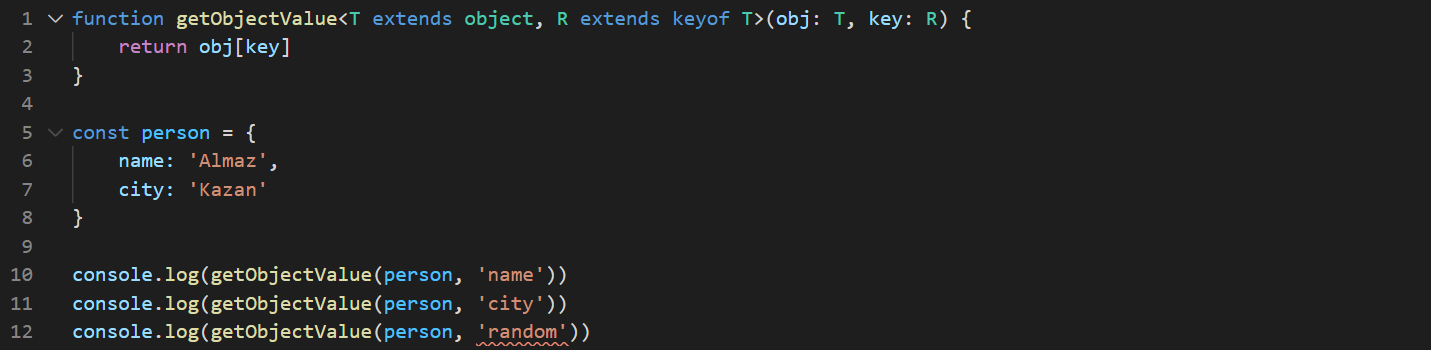
Однако на данный момент наша функция имеет недостаток. Generic принимает любые типы данных. Но мы хотим передавать только объект

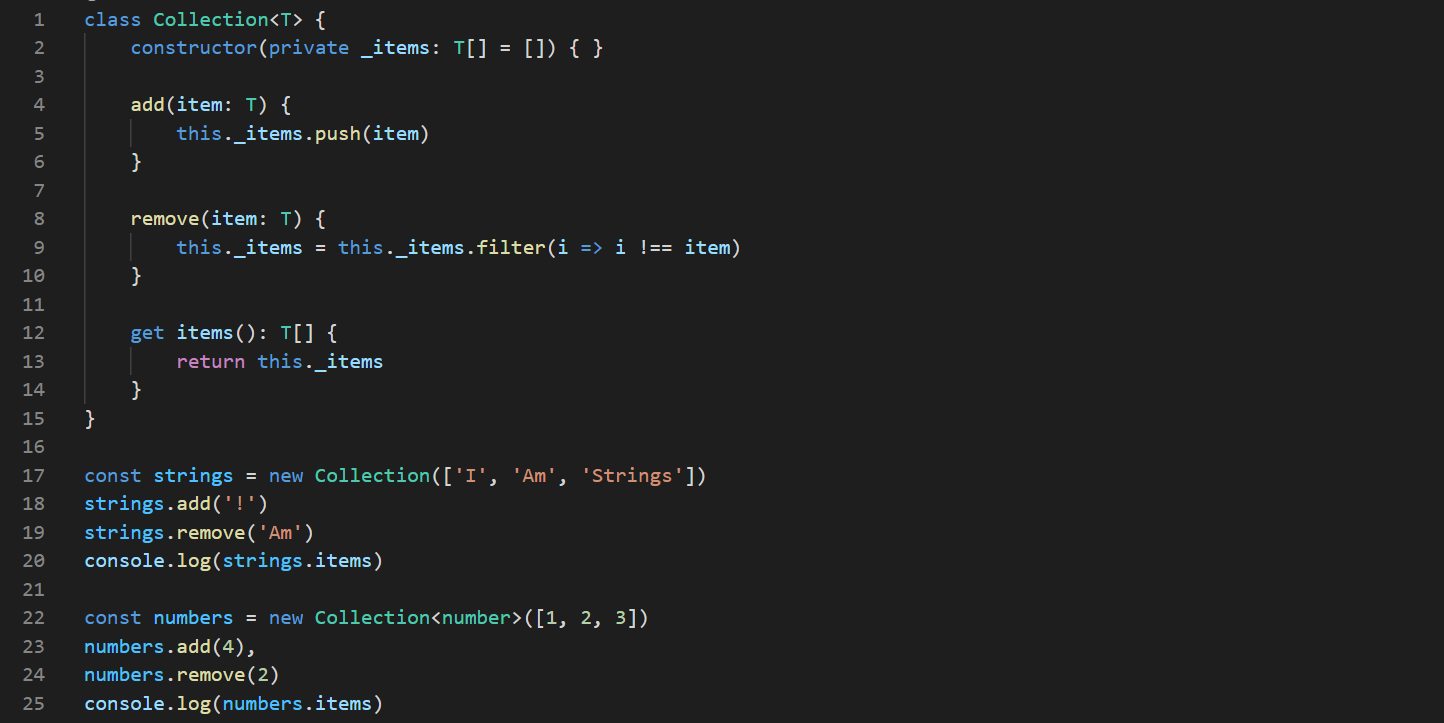
Сделаем, чтобы T наследовался от объекта. Теперь наша функция работает правильнее. Сразу видим то, что TypeScript подчеркивает строку 

**Рассмотрим еще один пример:**  
В данном примере мы создаем функцию, которая принимает строку, возвращает строку с учетом количества ее символов. Все хорошо, однако мы понимаем что метод length неопознан  


Для этого создадим интерфейс, теперь length не подсвечивается Запускаем файл, видим нужный результат

**Рассмотрим еще один пример:**  
В данном примере мы видим, что TypeScript не понимает о поле random

Чтобы исправить эту ошибку, мы превращаем функцию в Generic, причем делаем строку зависимой от объекта

**Generic на примере классов:**

**21.08.2020  
3 часа  
Отчет: Decorators TypeScript**

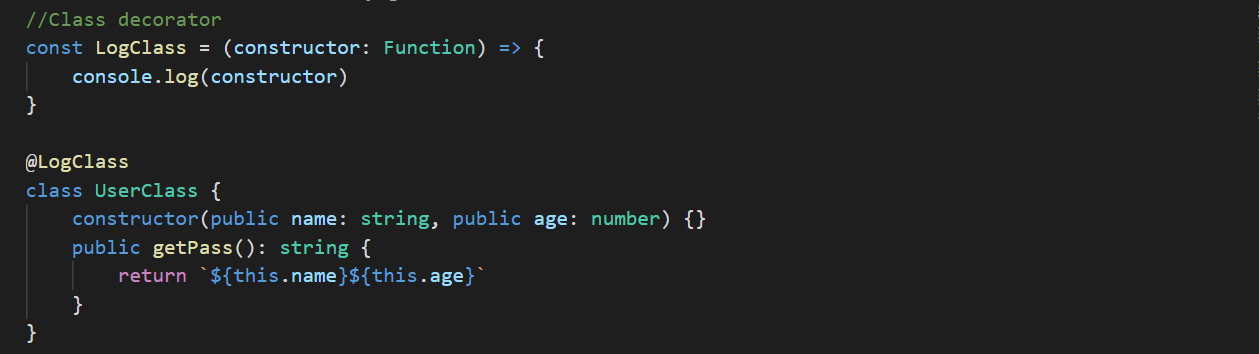
**Декораторы**Декораторы – это инструмент, который ползволяет добавить метаданные к классам и их членам, и тем самым они могут изменять поведение классов без изменения их кода  
В основном декораторы работают с классами  

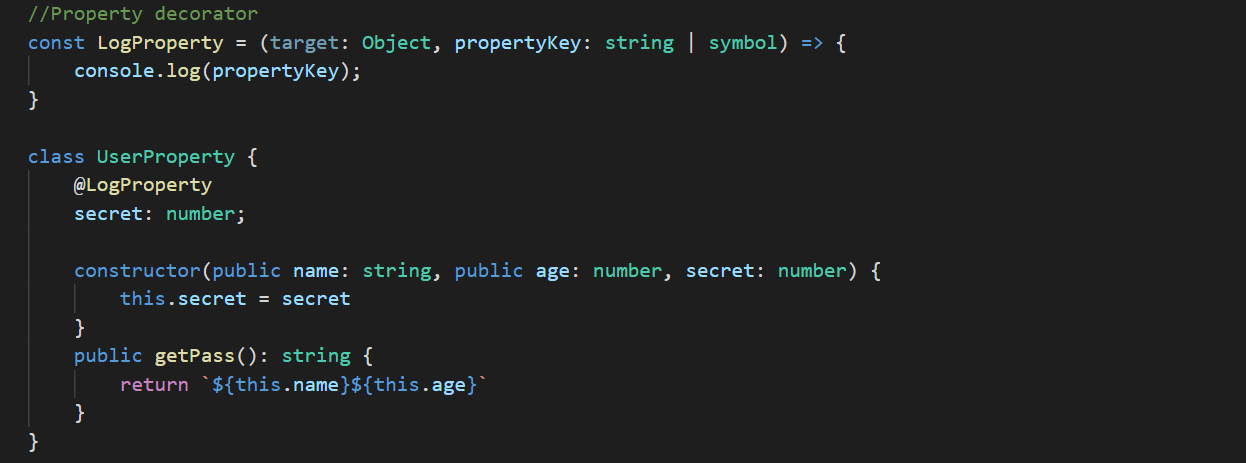

# **28.08.2020 4 часа Отчет: TypeScript – decorators, enums**

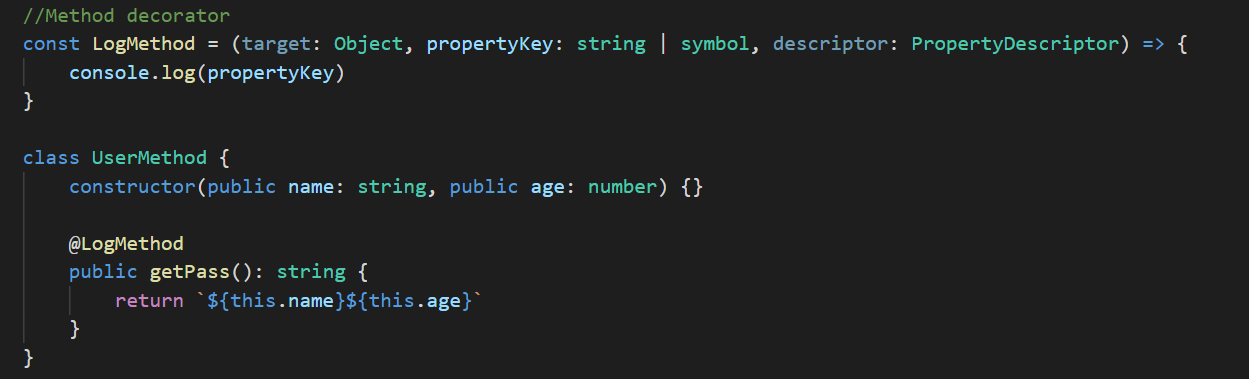
Декораторы по синтаксису – обычные функции

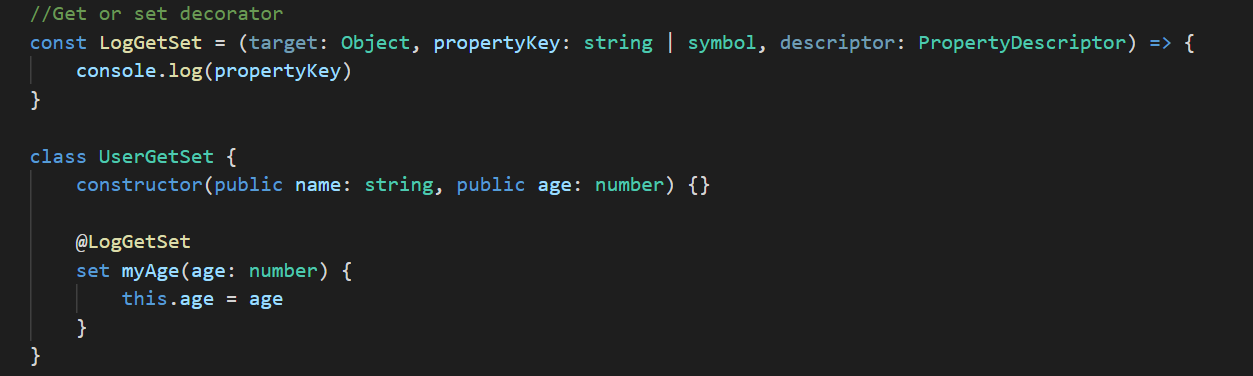
Правило декораторов: функция классового декоратора должна принимать конструктор декорируемой сущности

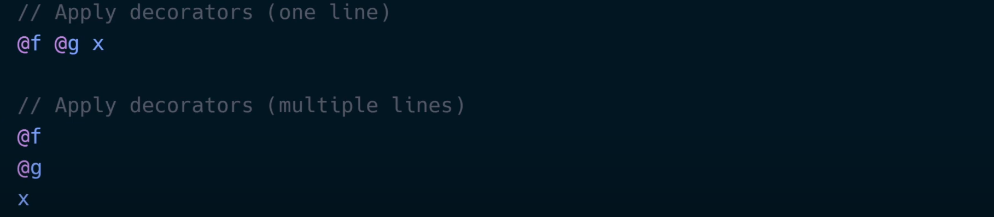
Существует 4 вида декораторов: декоратор класса, свойства, метода, аксессора

**Классовый декоратор:**

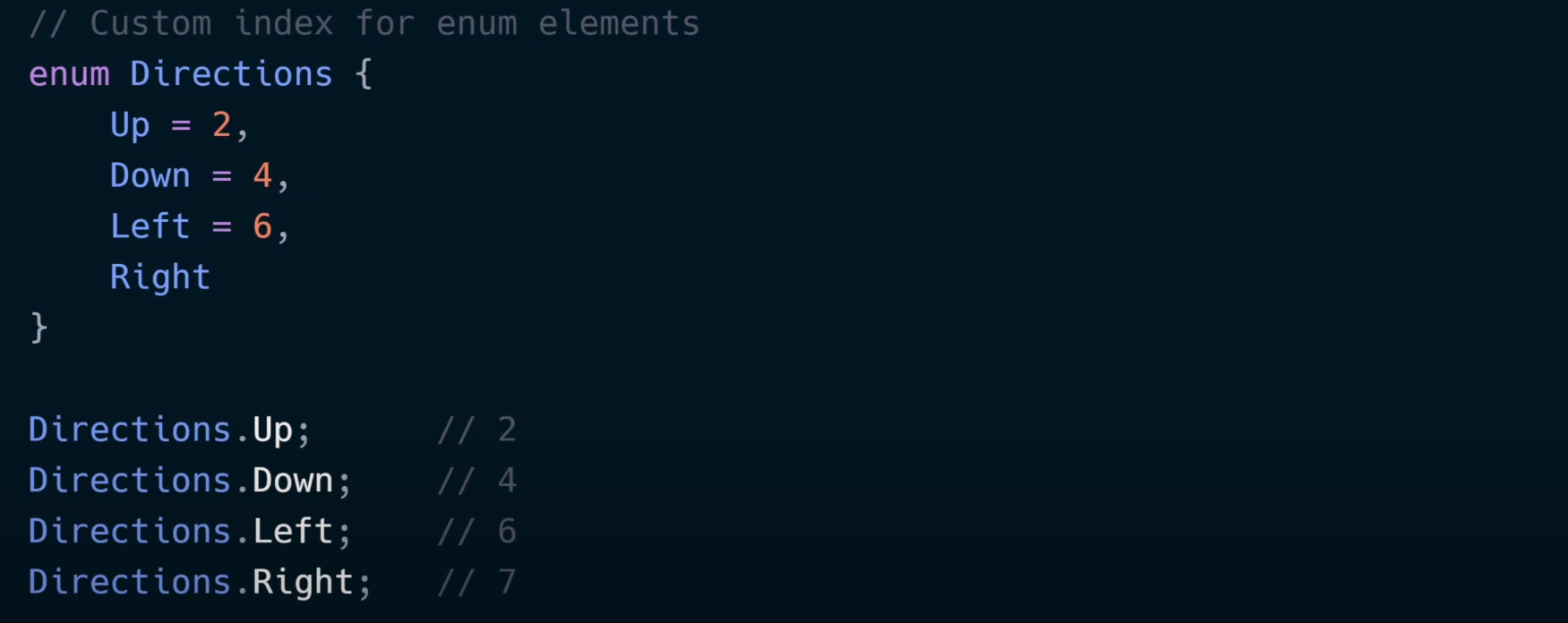
**Декоратор свойства:**

**Декоратор метода:**   


**Декоратор аксессора:**

**Композиция декораторов:**Также к одному объекту можно применять несколько декораторов. Существует два варианта записи:  


**Пример фабрики декораторов + композиции декораторов:**  ****

**Интересный пример с enum:**  


**Обращение к типу данных enum:**