

TypeScript In-Depth

Contents

| | |
|---|----|
| TypeScript In-Depth | 1 |
| 02. Types Basics..... | 3 |
| Task 02.01. Basic Types | 3 |
| Task 02.02. Const Assertions..... | 4 |
| 03. Functions | 5 |
| Task 03.01. Function Type..... | 5 |
| Task 03.02. Optional, Default and Rest Parameters | 6 |
| Task 03.03. Function Overloading..... | 7 |
| Task 03.04. Assertion Functions..... | 8 |
| 04. Interfaces..... | 9 |
| Task 04.01. Defining an Interface..... | 9 |
| Task 04.02. Defining an Interface for Function Types..... | 10 |
| Task 04.03. Extending Interface | 11 |
| Task 04.04. Optional Chaining..... | 12 |
| Task 04.05. Keyof Operator..... | 13 |
| 05. Classes | 14 |
| Task 05.01. Creating and Using Classes..... | 14 |
| Task 05.02. Extending Classes | 15 |
| Task 05.03. Creating Abstract Classes | 16 |
| Task 05.04. Interfaces for Class Types..... | 17 |
| Task 05.05. Intersection and Union Types | 18 |
| 06. Modules and Namespaces | 19 |
| Task 06.01. Using Namespaces | 19 |
| Task 06.02. Export and Import..... | 20 |
| Task 06.03. Default Export | 21 |
| Task 06.04. Re-Export..... | 22 |
| Task 06.05. Dynamic Import Expression | 23 |
| Task 06.06. Type-Only Imports and Exports | 24 |
| 07. Generics..... | 25 |

| | |
|--|----|
| Task 07.01. Generic Functions | 25 |
| Task 07.02. Generic Interfaces and Classes | 26 |
| Task 07.03. Generic Constraints..... | 27 |
| Task 07.04. Utility Types..... | 28 |
| 08. Decorators..... | 29 |
| Task 08.01. Class Decorators (sealed)..... | 29 |
| Task 08.02. Class Decorators that replace constructor functions (logger) | 30 |
| Task 08.03. Method Decorator (writable)..... | 31 |
| Task 08.04. Method Decorator (timeout) | 32 |
| Task 08.05. Parameter Decorator (logParameter) | 33 |
| Task 08.06. Property Decorator | 34 |
| Task 08.07. Accessor Decorator | 35 |
| 09. Asynchronous Patterns | 36 |
| Task 09.01. Callback Functions..... | 36 |
| Task 09.02. Promises..... | 37 |
| Task 09.03. Async Functions..... | 38 |

02. Types Basics

Task 02.01. Basic Types

1. Реализуйте функцию **getAllBooks()**, которая возвращает коллекцию книжек. Объявите эту коллекцию внутри функции, используя `let/const`.
 - a.

```
[
  { id: 1, title: 'Refactoring JavaScript', author: 'Evan Burchard', available: true },
  { id: 2, title: 'JavaScript Testing', author: 'Liang Yuxian Eugene', available: false },
  { id: 3, title: 'CSS Secrets', author: 'Lea Verou', available: true },
  { id: 4, title: 'Mastering JavaScript Object-Oriented Programming', author: 'Andrea Chiarelli', available: true }
]
```
2. Реализуйте функцию **logFirstAvailable()**, которая принимает массив книг в качестве параметра и выводит в консоль:
 - a. количество книг в массиве
 - b. название первой доступной книги
3. Запустите функцию **logFirstAvailable()**
4. Объявите **enum Category** для хранения следующих категорий книг:
 - a. JavaScript
 - b. CSS
 - c. HTML
 - d. TypeScript
 - e. Angular
5. Добавьте категорию к объектам в функции **getAllBooks()**
6. Реализуйте функцию **getBookTitlesByCategory()**, которая на вход будет получать категорию и возвращать массив наименований книг, которые принадлежат указанной категории. Используйте тип `Array<string>` и объявленный `enum`.
7. Реализуйте функцию **logBookTitles()**, которая принимает массив строк и выводит его в консоль. Используйте типы: `string[]` и `void`. Вызовите функции **getBookTitlesByCategory()** и **logBookTitles()**.
8. Реализуйте функцию **getBookAuthorByIndex()**, которая принимает `index` книжки в массиве и возвращает пару: название книжки + автор. Используйте `tuple` для возвращаемого типа. Вызовите данную функцию.
9. Реализуйте функцию **calcTotalPages()**, которая подсчитывает количество страниц книг в трех библиотеках города, используя следующие данные:
 - a.

```
[{ lib: 'libName1', books: 1_000_000_000, avgPagesPerBook: 250 }, { lib: 'libName2', books: 5_000_000_000, avgPagesPerBook: 300 }, { lib: 'libName3', books: 3_000_000_000, avgPagesPerBook: 280 }];
```
 - b. Для подсчетов используйте тип `BigInt`

Task 02.02. Const Assertions

1. Добавьте **const assertions** (<const>) для массива книг и массива, который предоставляет информацию о страницах книг в библиотеках города.
2. Добавьте модификатор **readonly** для параметра функции **logFirstAvailable()**

03. Functions

Task 03.01. Function Type

1. Создайте функцию **createCustomerID()**, которая принимает имя клиента (`name: string`) и его идентификатор (`id: number`) и возвращает конкатенацию этих значений в виде строки.
2. Объявите переменную **myID** строчного типа и вызовите функцию с значениями Ann, 10. Полученное значение выведите в консоль.
3. Объявите переменную **idGenerator** и задайте тип функции **createCustomerID()**. Присвойте этой переменной функциональное выражение, используя стрелочную функцию. Тело аналогично функции **createCustomerID()**.
4. Присвойте переменной **idGenerator** функцию **createCustomerID()** и вызовите ее. Полученное значение выведите в консоль.

Task 03.02. Optional, Default and Rest Parameters

1. Создайте функцию **createCustomer()**, которая принимает три параметра:
 - a. name: string – обязательный
 - b. age: number – необязательный
 - c. city: string – необязательный

Функция должна выводить имя клиента в лог используя template string, а также, если задан возраст, то она должна дополнительно выводить возраст в консоль. Если задан город, то дополнительно должна выводить город в консоль. Вызовите эту функцию с одним, двумя и тремя параметрами.

2. Внесите изменения в функцию **getBookTitlesByCategory()** – добавьте для параметра значение по умолчанию **Category.Javascript**, измените тело функции – замените for-of на forEach. Вызовите эту функцию без параметра.
3. Внесите изменения в функцию **logFirstAvailable()** – добавьте для параметра значение по умолчанию – вызов функции **getAllBooks()**. Вызовите эту функцию без параметра.
4. Создайте функцию **getBookById()**, которая принимает **id** книжки и возвращает книжку. Используйте функцию **getAllBooks()**, метод массива **find()** и стрелочную функцию. Вызовите функцию и передайте 1.
5. Создайте функцию **checkoutBooks()**, которая принимает два параметра:
 - a. customer: string
 - b. bookIds: number[] – переменное значение идентификаторов книжек

Функция должна проверить доступность каждой книжки, заданной идентификатором и вернуть массив наименований (title) книжек, которые доступны. (book.available = true). Используйте функцию **getBookById()**. Также функция должна выводить в лог имя заданного клиента.

6. Объявите переменную **myBooks** и сохраните в нее результат вызова функции **checkoutBooks('Ann', 1, 2, 4)**. Выведите результат в консоль.

Task 03.03. Function Overloading

1. Добавьте в первой строчке app.ts опцию для ESLint
`/* eslint-disable no-redeclare */`
Эта опция необходима для объявления нескольких сигнатур функций с одинаковыми именами
2. Создайте функцию **getTitles()**, которая может принимать 1 или 2 параметра:
 - a. если функция принимает 1 параметр, то он может быть либо string (author), либо boolean (available)
 - b. если функция принимает 2 параметра, то они должны быть id, available.

Функция должна возвращать массив книг по автору, или по доступности, или по id и доступности.

Для реализации функции создайте три сигнатуры с разными типами параметров и реализацию с rest параметром типа any[].

Функция должна анализировать количество и типы параметров с помощью оператора **typeof** и формировать результирующий массив из массива, полученного с помощью функции **getAllBooks()**, анализируя или свойство **book.author** или **book.available**.

3. Объявите переменную **checkedOutBooks** и вызовите функцию **getTitles(false)**. Выведите результат в консоль.

Task 03.04. Assertion Functions

1. Создайте функцию-утверждение **assertStringValue()**, которая принимает один параметр типа `any`. Функция должна проверять, является ли тип переданного аргумента строкой. Если нет, то генерировать исключение «**value should have been a string**».
2. Создайте функцию **bookTitleTransform()**, которая принимает один параметр - название книжки (тип параметра `any`). С помощью `assertStringValue` проверяет, действительно ли название книжки является строкой, и если да, то возвращает перевернуть этой строки, используя spread оператор и методы массива **reverse()** и **join()**.
3. Вызовите функцию **bookTitleTransform()** два раза и передайте ей строчное и числовое значение.

04. Interfaces

Task 04.01. Defining an Interface

1. Объявите интерфейс **Book**, который включает следующие поля:
 - a. `id` - число
 - b. `title` - строка
 - c. `author` - строка
 - d. `available` - логический
 - e. `category` – категория
2. Внесите изменения в функцию **getAllBooks()**, укажите тип, переменной **books** и тип возвращаемого значения, используя объявленный выше интерфейс **Book**. Добавьте модификатор **readonly**. Удалите временно `id` у книжки и увидите, что появится ошибка.
3. Внесите изменения в функцию **getBookById()**, укажите тип возвращаемого значения, используя объявленный выше интерфейс. Возможно, понадобится добавить объединение с типом **undefined**, поскольку метод **find**, если не найдет элемент, вернет **undefined**.
4. Создайте функцию **printBook()**, которая на вход принимает книгу и выводит в консоль фразу **book.title + by + book.author**. Для типа параметра используйте интерфейс **Book**.
5. Объявите переменную **myBook** и присвойте ей следующий объект

```
{
  id: 5,
  title: 'Colors, Backgrounds, and Gradients',
  author: 'Eric A. Meyer',
  available: true,
  category: Category.CSS,
  year: 2015,
  copies: 3
}
```
6. Вызовите функцию **printBook()** и передайте ей **myBook**. Никаких ошибок при этом не должно появляться.
7. Добавьте в интерфейс **Book** свойство **pages: number**. Вы получите ошибку в функции **getAllBooks()**. Чтобы ошибка не возникала сделайте свойство не обязательным.
8. Укажите явно для переменной **myBook** тип **Book**. Вы снова получите ошибку. Удалите свойства **year, copies**. Добавьте свойство **pages: 200**.
9. Добавьте в интерфейс **Book** необязательное свойство **markDamaged**, которое является методом. Метод принимает на вход строчный параметр **reason** и ничего не возвращает. Добавьте этот метод в объект **myBook**. Метод должен выводить строку **`Damaged: \${reason}`**, используя стрелочную функцию. Вызовите этот метод и передайте строку **`missing back cover`**

Task 04.02. Defining an Interface for Function Types

1. Объявите интерфейс **DamageLogger**, который будет описывать тип для функции, которая принимает один строчный параметр и ничего не возвращает.
2. Внесите изменения в интерфейс **Book**: используйте объявленный интерфейс для поля **markDamaged**.
3. Объявите переменную **logDamage** используя объявленный ранее интерфейс. Создайте функцию, которая удовлетворяет этому интерфейсу, присвойте объявленной переменной. Вызовите функцию.

Task 04.03. Extending Interface

1. Объявите интерфейс **Person**, который содержит два строчных свойства – **name** и **email**.
2. Объявите интерфейс **Author** на основе интерфейса **Person**, который расширяет указанный интерфейс числовым свойством **numBooksPublished**.
3. Объявите интерфейс **Librarian** на основе интерфейса **Person**, который расширяет указанный интерфейс двумя свойствами:
 - a. Строчное свойство **department**
 - b. Функция **assistCustomer**, которая принимает строчный параметр **custName** и ничего не возвращает.
4. Объявите переменную **favoriteAuthor** используя интерфейс **Author**, задайте значение в виде литерала объекта.
5. Объявите переменную **favoriteLibrarian** используя интерфейс **Librarian**, задайте значение в виде литерала объекта.

Task 04.04. Optional Chaining

1. Объявите переменную `offer` следующего вида:

```
const offer: any = {  
  book: {  
    title: 'Essential TypeScript',  
  },  
};
```

2. Выведите в консоль значение следующих выражений, используя optional chaining.
 - a. `offer.magazine`
 - b. `offer.magazine.getTitle()`
 - c. `offer.book.getTitle()`
 - d. `offer.book.authors[0]`

Task 04.05. Keyof Operator

1. Объявите тип **BookProperties**, который включает свойства интерфейса **Book**, используя **keyof** оператор.
2. Реализуйте функцию **getProperty()**, которая принимает два параметра:
 - a. книжку
 - b. название свойства из интерфейса **Book**и возвращает значение этого свойства из переданного объекта, если это не функция, для функции возвращает ее имя. Используйте тип **any** для возвращаемого значения.
3. Вызовите эту функцию три раза со значением для второго параметра: **title**, **markDamaged**, **isbn**.

05. Classes

Task 05.01. Creating and Using Classes

1. Создайте класс **Referenceltem**, который содержит:
 - a. Строчное свойство **title**
 - b. Числовое свойство **year**
 - c. Конструктор с двумя параметрами: строчный параметр **newTitle**, числовой параметр **newYear**, который в консоль выводит строку 'Creating a new Referenceltem...' и инициализирует поля.
 - d. Метод **printItem()** без параметров, который ничего не возвращает. Этот метод должен использовать template string literal и выводить строку «**title was published in year**» в консоль.
2. Объявите переменную **ref** и проинициализируйте ее объектом **Referenceltem**. Передайте значения параметров в конструктор. Вызовите метод **printItem()**.
3. Закомментируйте конструктор, свойства **title** и **year** и реализуйте создание свойств через параметры конструктора (**title- public, year - private**).
4. Создайте приватное ("soft private") строчное свойство **_publisher**.
 - a. Добавьте геттер **publisher**, который преобразовывает свойство **_publisher** в верхний регистр и возвращает его.
 - b. Добавьте сеттер **publisher**, который принимает строчный параметр **newPublisher** и устанавливает значение свойства **_publisher** в значение этого параметра.
 - c. Проинициализируйте свойство **ref.publisher** каким-либо строчным значением и выведите его в консоль. Результат должен быть в верхнем регистре.
5. Создайте приватное ("hard private") числовое свойство **id**.
 - a. Внесите изменения в конструктор для инициализации этого свойства.
 - b. Добавьте метод **getID()**, который должен возвращать значение свойства **id**.
 - c. Выведите объект в консоль.
 - d. Вызовите метод **getID()**.
6. Создайте статичное строчное свойство **department** и проинициализируйте его каким-либо значением по умолчанию. Внесите изменения в метод **printItem()** – добавьте вывод в консоль этого статического свойства.

Task 05.02. Extending Classes

1. Создайте класс **Encyclopedia** как наследника класса **Referenceltem**. Добавьте одно дополнительное числовое публичное свойство **edition**. Используйте параметры конструктора.
2. Объявите переменную **refBook** и создайте объект **Encyclopedia**. Вызовите метод **printItem()**;
3. Переопределите метод **printItem()**. Пусть он делает то, что делал и дополнительно выводит строку в консоль «**Edition: edition (year)**». Вы получите ошибку, что свойство **year** недоступно. Чтобы оно было доступно измените модификатор доступа в классе **Referenceltem** на **protected**.

Task 05.03. Creating Abstract Classes

1. Внесите изменения в класс **ReferenceItem** – сделайте его абстрактным.
2. Добавьте абстрактный метод **printCitation()**, который не принимает параметров и не возвращает значения. У этого метода не должно быть реализации. После этого Вы получите ошибку в классе **Encyclopedia**, которая будет сообщать, что не реализован абстрактный метод.
3. Добавьте реализацию метода **printCitation** в класс **Encyclopedia**. Метод должен выводить в консоль строку «**title – year**».
4. Объявите переменную **refBook** и создайте объект **Encyclopedia**. Вызовите метод **printCitation()**;

Task 05.04. Interfaces for Class Types

1. Создайте класс **UniversityLibrarian**, который реализует интерфейс **Librarian** и реализуйте все необходимые свойства. Метод **assistCustomer** должен выводить в консоль строку ``${this.name} is assisting ${custName}``.
2. Объявите переменную **favoriteLibrarian** используя интерфейс **Librarian** и проинициализируйте ее с помощью объекта, созданного классом **UniversityLibrarian()**. Никаких ошибок при этом не должно возникать. Проинициализируйте свойство **name** и вызовите метод **assistCustomer()**.

Task 05.05. Intersection and Union Types

1. Создайте тип **PersonBook**. Используйте для этого интерфейсы **Person**, **Book** и пересечение типов.
2. Объявите переменную с типом **PersonBook**, проинициализируйте ее литералом, выведите ее в консоль.
3. Создайте тип **BookOrUndefined**. Используйте для этого объединение интерфейса **Book** и **undefined**.
4. Замените тип возвращаемого значения в функции **getBookByID** на **BookOrUndefined**.

06. Modules and Namespaces

Task 06.01. Using Namespaces

1. Создайте папку для нового проекта **NamespaceDemo**
2. Создайте файл **utility-functions.ts**
3. Создайте пространство имен **Utility**
4. Создайте вложенное пространство имен **Fees**
5. Создайте и экспортируйте функцию **calculateLateFee()** во вложенном пространстве имен, которая принимает числовой параметр **daysLate** и возвращает **fee**, вычисленное как **daysLate * 0.25**;
6. Создайте и экспортируйте функцию **maxBooksAllowed()** в пространстве имен **Utility**, которая принимает один числовой параметр **age**. Если **age < 12**, то возвращает **3** иначе **10**.
7. Создайте функцию **privateFunc()**, которая выводит в консоль сообщение «**This is private function**»
8. Создайте файл **app.ts**. Добавьте ссылку на файл **utility-functions.ts**
9. Напишите фрагмент кода, который использует функции из пространства имен.
10. Используйте ключевое слово **import** и объявите алиас **util** для вложенного пространства имен.
import util = Utility.Fees;
11. Запустите компилятор и скомпилируйте только **tsc app.ts --target ES5**. Создайте **index.html**
Воспользуйтесь следующим фрагментом HTML:

```
<html>
  <head></head>
  <body>
    <script src="utility-functions.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```
12. Запустите еще раз компилятор и укажите опцию **--outFile bundle.js**
13. Подключите полученный файл в **index.html**

Task 06.02. Export and Import

1. Создайте файл **enums.ts**, перенесите в него **enum Category**. Добавьте экспорт в конце файла.
2. Создайте файл **intefaces.ts** и перенесите в него интерфейсы:
 - a. **Book, DamageLogger, Person, Author, Librarian**
 - b. Добавьте импорт **Category**
 - c. Добавьте экспорт интерфейсов **Book, DamageLogger, Person, Author, Librarian** в конце файла. Экспортируйте **DamageLogger** с именем **Logger**
3. Создайте новый файл **classes.ts** и перенесите в него классы. **UniversityLibrarian, Referenceltem**.
 - a. Добавьте импорт интерфейсов как целого модуля с именем **Interfaces**
 - b. Измените описание класса **UniversityLibrarian**, чтобы он реализовывал интерфейс **Interfaces.Librarian**
 - c. Добавьте экспорт в конце файла и экспортируйте оба класса.
4. Создайте файл **types.ts** и перенесите в него типы: **BookProperties, PersonBook, BookOrUndefined**.
 - a. Добавьте импорт интерфейсов **Book** и **Person**
 - b. Экспортируйте типы из модуля.
5. Создайте файл **functions.ts** и перенесите все функции.
 - a. Добавьте импорт интерфейса **Book**, перечисления **Category**, типов **BookProperties, BookOrUndefined**
 - b. Добавьте экспорт для всех функций (не обязательно)
6. Внесите изменения в файл **app.ts**
 - a. Добавьте импорт **Category**, интерфейсов **Book, Logger, Author, Librarian**, классов **UniversityLibrarian, Referenceltem**, тип **PersonBook** и всех функций.
 - b. Измените тип переменной **logDamage** на **Logger (Task 04.02)**

Task 06.03. Default Export

1. Создайте файл **encyclopedia.ts** и переместите в него класс **Encyclopedia**. Добавьте импорт **Referenceltem**. Добавьте экспорт по умолчанию.
2. Импортируйте данный класс в приложение как **RefBook**
3. Внесите изменения в код задания 05.02.

Task 06.04. Re-Export

1. Создайте папку **classes** и переместите в нее файл **encyclopedia.ts**
2. Разнесите классы **UniversityLibrarian** и **ReferenceItem** по разным файлам и тоже переместите в папку **classes**.
3. Удалите файл **classes.ts**
4. Создайте файл **classes/index.ts** и добавьте в него реэкспорт классов **Encyclopedia**, **ReferenceItem**, **UniversityLibrarian**.
5. Исправьте импорты в файле **app.ts**

Task 06.05. Dynamic Import Expression

1. Создайте в папке **classes** файл **reader.ts** и реализуйте класс `Reader`, который содержит следующие свойства:
 - a. `name: string;`
 - b. `books: Book[] = [];`
 - c. `take(book: Book): void` - метод добавляет книжку в массив книжек.
2. Внесите изменения в файл **classes/index.ts**, добавьте новый модуль.
3. Реализуйте выражение динамического импорта, для загрузки всего из пути `'./classes'` как модуля. Создайте экземпляр класса `Reader`. Выведите его в консоль.

Task 06.06. Type-Only Imports and Exports

1. Создайте в папке **classes** файл **library.ts** и реализуйте класс **Library**, который содержит следующие свойства:
 - a. `Id: number`
 - b. `name: string`
 - c. `address: string`
2. Внесите изменения в файл **classes/index.ts**. Экспортируйте тип **Library**. Используйте конструкцию `export type {...}`
3. Импортируйте тип **Library** в приложение. Объявите переменную, используя тип **Library**.
4. Создайте экземпляр класса **Library**. Вы должны получить ошибку. Закомментируйте строку.
5. Объявите переменную, укажите тип `Library`. Проинициализируйте литералом, выведите в консоль.

07. Generics

Task 07.01. Generic Functions

1. Создайте в файле **functions.ts** дженерик функцию **purge()**, которая принимает один параметр – дженерик массив **inventory** и возвращает дженерик массив того же типа, который содержит элементы первоначального массива без двух первых элементов. Экспортируйте данную функцию.
2. Импортируйте данную функцию в приложение.
3. Добавьте категорию **Software** в файле **enums.ts**.
4. Объявите переменную **inventory**, которая содержит следующий массив книг

```
[  
{ id: 10, title: 'The C Programming Language', author: 'K & R', available: true, category: Category.Software },  
{ id: 11, title: 'Code Complete', author: 'Steve McConnell', available: true, category: Category.Software },  
{ id: 12, title: '8-Bit Graphics with Cobol', author: 'A. B.', available: true, category: Category.Software },  
{ id: 13, title: 'Cool autoexec.bat Scripts!', author: 'C. D.', available: true, category: Category.Software }  
];
```

5. Вызовите функцию **purge()** и передайте ей эти данные. Выведите результат в консоль.
6. Вызовите эту же функцию, но с числовым массивом и снова выведите результат в консоль.

Task 07.02. Generic Interfaces and Classes

1. Создайте интерфейс **Magazine**, который содержит два строчных свойства **title**, **publisher** и добавьте его в файл **interfaces.ts**. Экспортируйте данный интерфейс.
2. Создайте файл **classes/shelf.ts** и используя экспорт по умолчанию реализуйте дженерик класс **Shelf**:
 - a. добавьте приватное свойство **items**, которое является массивом элементов типа **T**.
 - b. добавьте метод **add()**, который принимает один параметр **item** типа **T** и добавляет его в массив. Ничего не возвращает.
 - c. добавьте метод **getFirst()**, который ничего не принимает, а возвращает первый элемент с полки.
3. Добавьте реэкспорт в файл **classes/index.ts**
4. Импортируйте данный класс и интерфейс **Magazine** в приложение.
5. Закомментируйте код, который относится к функции **purge()**, кроме переменной **inventory**
6. Создайте полку **bookShelf** и сохраните все книжки из **inventory** на полку. Получите первую книжку и выведите ее название в консоль.
7. Объявите переменную **magazines**, которая содержит следующие данные:

```
[
  { title: 'Programming Language Monthly', publisher: 'Code Mags' },
  { title: 'Literary Fiction Quarterly', publisher: 'College Press' },
  { title: 'Five Points', publisher: 'GSU' }
];
```
8. Создайте полку **magazineShelf** и поместите все эти журналы на полку. Получите первый журнал и выведите его в консоль.

Task 07.03. Generic Constraints

1. Внесите изменения в класс **Shelf**:
 - a. добавьте метод **find()**, который принимает строчный параметр **title** и возвращает первый найденный элемент на полке типа **T**.
 - b. добавьте метод **printTitles()**, который выводит в консоль наименования того, что находится на полке.После добавления этих методов вы должны получить ошибку, что свойство **title** не существует.
2. В файле **interfaces.ts** создайте интерфейс **ShelfItem**, который должен содержать все необходимые свойства, которые должен иметь тип **T**, а именно **title**.
3. Добавьте дженерик ограничение для класса расширив тип **T**.
4. Вызовите функцию **printTitles()** для журналов.
5. Найдите журнал **'Five Points'** и выведите его в консоль.
6. Внесите изменения в функцию **getProperty**. Добавьте два параметра типа **TObject**, **TKey**. Добавьте ограничение на второй параметр, чтобы значения были только ключами объекта типа **TObject**, используя **keyof** оператор. Измените тип возвращаемого значения на **TObject[TKey] | string**. Вызовите эту функцию.

Task 07.04. Utility Types

1. Объявите алиас типа **BookRequiredFields** в файле **types.ts**, используя интерфейс **Book** и утилиту **Required**.
2. Объявите переменную типа **BookRequiredFields** и присвойте ей соответствующий объект.
3. Объявите алиас типа **UpdatedBook**, используя интерфейс **Book** и утилиту **Partial**
4. Объявите переменную типа **UpdatedBook** и присвойте ей соответствующий объект.
5. Объявите алиас типа **AuthorWoEmail**, используя интерфейс **Author** и утилиту **Omit**.
6. Объявите алиас **CreateCustomerFunctionType** для функционального типа функции **createCustomer**.
7. Объявите переменную используя алиас типа **CreateCustomerFunctionType** и утилиту **Parameters**, вызовите функцию **createCustomer**, передав эту переменную

08. Decorators

Task 08.01. Class Decorators (sealed)

1. Создайте файл **decorators.ts**. Создайте декоратор класса **@sealed()**, для того, чтобы предотвратить добавление новых свойств объекту класса и прототипу объекта. Функция-декоратор должна принимать один строчный параметр и ничего не должна возвращать. Перед выполнением функционала функция должна вывести в консоль сообщение **«Sealing the constructor + параметр»**. Используйте метод **Object.seal()**.
2. Примените данный декоратор к классу **UniversityLibrarian**.
3. Создайте экземпляр класса **UniversityLibrarian**. Проверьте сообщение в консоли.

Task 08.02. Class Decorators that replace constructor functions (logger)

1. Создайте декоратор класса **@logger()**, который будет изменять конструктор класса.
2. Объявите внутри декоратора переменную **newConstructor: Function** и проинициализируйте ее функциональным выражением. Новый конструктор должен
 - a. выводить в консоль сообщение «**Creating new instance**»
 - b. выводить переданный параметр (имя класса).
 - c. создавать новое свойство **age** со значением **30**.
3. Проинициализируйте прототип нового конструктора объектом, созданным на основе прототипа переданного класса используя **Object.create()**.
4. Добавьте новый метод в прототип нового конструктора **printLibrarian()**, который должен выводить в консоль **`Librarian name: \${this.name}, Librarian age: \${this.age}`**.
5. Верните из декоратора новый конструктор, предварительно преобразовав его к типу **<TFunction>**.
6. Примените этот декоратор к классу **UniversityLibrarian**. Проверьте результат работы в консоли.
7. Объявите переменную **fLibrarian** и создайте экземпляр класса **UniversityLibrarian**. Задайте значение **Anna** для **name**. Вызовите метод **printLibrarian()**

Task 08.03. Method Decorator (writable)

1. Создайте декоратор метода **@writable()** как фабрику, которая получает булевый параметр **isWritable**. Декоратор должен устанавливать свойство дескриптора **writable** в переданное значение.
2. Добавить два метода для класса **UniversityLibrarian**
 - a. **assistFaculty()** – выводит в консоль сообщение «**Assisting faculty**».
 - b. **teachCommunity()** – выводит в консоль сообщение «**Teaching community**».
3. Задекорируйте метод **assistFaculty()** как изменяемый, а метод **teachCommunity()** неизменяемый.
4. Попробуйте поменять методы у экземпляра этого класса.

Task 08.04. Method Decorator (timeout)

1. Создать декоратор метода **@timeout()** как фабрику, которая получает числовой параметр – количество миллисекунд. Метод, к которому применяется декоратор, должен запускаться через указанное количество времени.
2. Декоратор должен переопределять свойство дескриптора **value**. Новая функция должна использовать **setTimeout()** и запускать первоначальный метод через указанное количество времени. Вернуть из декоратора новый дескриптор.
3. Применить декоратор к методу **printItem()** класса **ReferenceItem**.
4. Создайте экземпляр класса **Encyclopedia** и вызовите метод **printItem()**

Task 08.05. Parameter Decorator (logParameter)

1. Создайте декоратор параметра метода - **@logParameter()**, который должен сохранять индекс параметра, к которому применяется декоратор в свойство прототипа **\${methodName}_decor_params_indexes**. Свойство организовать в виде массива.
2. Создайте декоратор метода **@logMethod()**. Декоратор должен переопределять метод, к которому он применяется и возвращать новый дескриптор.
3. Переопределенный метод должен получить доступ к индексам, находящимся в свойстве **\${methodName}_decor_params_indexes** и для каждого параметра выводить его значение в формате **Method: \${methodName}, ParamIndex: \${ParamIndex}, ParamValue: \${ParamValue}**
4. Задекорируйте метод **assistCustomer()** и все его параметры соответствующими декораторами.
5. Создайте экземпляр класса **UniversityLibrarian**, проинициализируйте свойство **name**, вызовите метод **assistCustomer()**.

Task 08.06. Property Decorator

1. Создайте фабричную функцию декоратора свойства **@format(pref: string = 'Mr./Mrs.')**, которая при применении к свойству форматирует его вывод – добавляет префикс **pref**. Фабричная функция должна возвращать функцию с сигнатурой декоратора свойства, внутри которой необходимо вызвать функцию **makeProperty(target, propertyName, value => `\${pref} \${value}`, value => value)**;
2. Функция **makeProperty** имеет следующий вид:

```
function makeProperty<T>(  
  prototype: any,  
  propertyName: string,  
  getTransformer: (value: any) => T,  
  setTransformer: (value: any) => T  
) {  
  const values = new Map<any, T>();  
  
  Object.defineProperty(prototype, propertyName, {  
    set(firstValue: any) {  
      Object.defineProperty(this, propertyName, {  
        get() {  
          if (getTransformer) {  
            return getTransformer(values.get(this));  
          } else {  
            values.get(this);  
          }  
        },  
        set(value: any) {  
          if (setTransformer) {  
            values.set(this, setTransformer(value));  
          } else {  
            values.set(this, value);  
          }  
        },  
        enumerable: true  
      });  
      this[propertyName] = firstValue;  
    },  
    enumerable: true,  
    configurable: true  
  });  
}
```

3. Добавьте функцию **makeProperty** в **functions.ts**
4. Задекорируйте свойство **name** класса **UniversityLibrarian** декоратором **@format()**
5. Создайте экземпляр класса **UniversityLibrarian**. Установите значение для свойства **name**, затем получите его и выведите в консоль.

Task 08.07. Accessor Decorator

1. Создайте декоратор аксессора **@positiveInteger()**, который бросает исключение в случае, если свойству устанавливается значение меньше **1** и не целое.
2. Добавьте в класс **Encyclopedia** приватное числовое свойство **_copies**, а также **getter** и **setter** для этого свойства, которые возвращают значение и устанавливают значение соответственно.
3. Задекорируйте getter или setter декоратором **@positiveInteger()**.
4. Создайте экземпляр класса **Encyclopedia**. Попробуйте установить разные значения, **-10, 0, 4.5, 5**

09. Asynchronous Patterns

Task 09.01. Callback Functions

1. В файле **interfaces.ts** создайте интерфейс для функции обратного вызова **LibMgrCallback**, которая принимает два параметра:
 - a. **err: Error**,
 - b. **titles: string[]**и ничего не возвращает
2. В файле **functions.ts** создайте функцию **getBooksByCategory()**, которая принимает два параметра:
 - a. **category** - категории
 - b. **callback** – тип, ранее созданный интерфейси ничего не возвращает
3. Функция должна использовать **setTimeout()** и через 2с выполнить следующий код:
 - a. В секции **try**: Использовать функцию **getBookTitlesByCategory()** для получения заголовков книг по категории
 - b. Если нашли книги, то вызвать функцию обратного вызова и передать два параметра: **null** и найденные книги
 - c. Если не нашли книг, то бросить исключение **throw new Error('No books found.');**
 - d. В секции **catch**: вызвать функцию обратного вызова и передать два параметра **error** и **null**.
4. Создайте функцию **logCategorySearch()**, которая имеет сигнатуру, описанную в интерфейсе **LibMgrCallback**. Если пришел объект ошибки, то вывести свойство **err.message**, в противном случае вывести названия книг.
5. Вызовите функцию **getBooksByCategory()** и передайте ей необходимые аргументы. Добавьте вывод сообщений в консоль перед и после вызова этой функции. Используйте **Category.JavaScript** и **Category.Software** в качестве значения первого параметра.

Task 09.02. Promises

1. Создайте функцию **getBooksByCategoryPromise()**, которая принимает один параметр – **category** и возвращает промис – массив заголовков книг.
2. Используйте **new Promise((resolve, reject) => { setTimeout(() => {...}, 2000) });** Добавьте код, аналогичный функции **getBooksByCategory()**, только теперь используйте **resolve()** и **reject()**. Верните из функции созданный промис.
3. Вызовите функцию **getBooksByCategoryPromise()** и зарегистрируйте функции обратного вызова с помощью методов **then** и **catch**. Добавьте вывод сообщений в консоль перед и после вызова этой функции. Используйте **Category.JavaScript** и **Category.Software** в качестве значения параметра.
4. Верните из функции, зарегистрированной с помощью **then()**, количество найденных книг. Зарегистрируйте с помощью еще одного метода **then()** функцию, которая должна вывести в консоль количество найденных книг.

Task 09.03. Async Functions

1. Создайте асинхронную функцию **logSearchResults** в файле **functions.ts**. Функция должна использовать функцию **getBooksByCategoryPromise**, получать и выводить в консоль количество найденных книг
2. Вызовите эту функцию. Задайте значение параметра **Category.JavaScript**. Добавьте вывод в консоль до и после вызова функции. Обработайте ошибку с помощью **catch**