

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
Task 07.05. Mapped Types, Utility Types, Conditional Types	29
08. Decorators	30
Task 08.01. Class Decorators (sealed).....	30
Task 08.02. Class Decorators that replace constructor functions (logger).....	31
Task 08.03. Method Decorator (writable)	32
Task 08.04. Method Decorator (timeout)	33
Task 08.05. Parameter Decorator (logParameter).....	34
Task 08.06. Property Decorator	35
Task 08.07. Accessor Decorator	36
09. Asynchronous Patterns	37
Task 09.01. Callback Functions	37
Task 09.02. Promises.....	38
Task 09.03. Async Functions	39

02. Types Basics

Task 02.01. Basic Types

1. Реалізуйте функцію **getAllBooks()**, яка повертає колекцію книжок. Об'явіть цю колекцію всередині функції.

```
[
  { 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. Реалізуйте функцію **logFirstAvailable()**, яка приймає масив книг як параметр і виводить у консоль:
 - a. кількість книг у масиві
 - b. назву першої доступної книги
3. Запустіть функцію **logFirstAvailable()**
4. Об'явіть **enum Category** для зберігання наступних категорій книг:
 - a. JavaScript
 - b. CSS
 - c. HTML
 - d. TypeScript
 - e. Angular
5. Додайте категорію до об'єктів у функції **getAllBooks()**
6. Реалізуйте функцію **getBookTitlesByCategory()**, яка на вхід отримує категорію та повертає масив найменувань книг, що належать зазначеній категорії.
7. Реалізуйте функцію **logBookTitles()**, яка приймає масив рядків та виводить його в консоль. Викличте функції **getBookTitlesByCategory()** та **logBookTitles()**.
8. Реалізуйте функцію **getBookAuthorByIndex()**, яка приймає **index** книжки у масиві та повертає пару: назву книжки + автор. Використовуйте **tuple** для типу, що повертається. Викличте цю функцію.
9. Внесіть зміни до типу, що повертається функцією **getBookAuthorByIndex()** – додайте мітки: **title**, **author** для типу **tuple**
10. Реалізуйте функцію **calcTotalPages()**, яка підраховує кількість сторінок книг у трьох бібліотеках міста, використовуючи такі дані:

```
[
  { 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 }
];
```

Для підрахунків використовуйте тип **bigint**

Task 02.02. Const Assertions

1. Додайте **const assertions** (<const>) для масиву книг та масиву, який надає інформацію про сторінки книг у бібліотеках міста.
2. Додайте модифікатор **readonly** для параметра функції **logFirstAvailable()**

03. Functions

Task 03.01. Function Type

1. Створіть функцію **createCustomerID()**, яка приймає ім'я клієнта (`name: string`) та його ідентифікатор (`id: number`) та повертає конкатенацію цих значень у вигляді рядка.
2. Об'явіть змінну **myID** рядкового типу та викличте функцію зі значеннями App, 10. Отримане значення виведіть у консоль.
3. Об'явіть змінну **idGenerator** і вкажіть тип функції **createCustomerID()**. Надайте цій змінній функціональний вираз, використовуючи стрілочну функцію. Тіло подібне до функції **createCustomerID()**.
4. Надайте змінній **idGenerator** функцію **createCustomerID()** та викличте її. Отримане значення виведіть у консоль.

Task 03.02. Optional, Default and Rest Parameters

1. Створіть функцію **createCustomer()**, яка приймає три параметри:
 - a) name: string – обов'язковий
 - b) age: number – необов'язковий
 - c) city: string – необов'язковий

Функція повинна виводити ім'я клієнта в консоль, а також, якщо заданий вік, вона повинна додатково виводити вік у консоль. Якщо задане місто, то додатково має виводити місто у консоль. Викличте цю функцію з одним, двома та трьома аргументами.

2. Внесіть зміни до функції **getBookTitlesByCategory()** – додайте для параметра значення за замовчуванням **Category.Javascript**. Викличте цю функцію без аргументів.
3. Внесіть зміни до функції **logFirstAvailable()** – додайте для параметра значення за замовчуванням – виклик функції **getAllBooks()**. Викличте цю функцію без аргументів.
4. Створіть функцію **getBookById()**, яка приймає id книжки та повертає книжку. Використовуйте функцію **getAllBooks()**, метод масиву **find()** та стрілочну функцію. Викличте функцію та передайте їй 1.
5. Створіть функцію **checkoutBooks()**, яка приймає два параметри:
 - a. customer: string
 - b. bookIds: number[] – змінне значення ідентифікаторів книжок (рест параметр)

Функція повинна перевірити доступність кожної книжки, заданої ідентифікатором, та повернути масив найменувань (title) книжок, які є доступними. (available = true).

Використовуйте функцію **getBookById()**. Також функція повинна виводити в консоль ім'я заданого клієнта.

6. Об'явіть змінну **myBooks** та збережіть у ній результат виклику функції **checkoutBooks('Ann', 1, 2, 4)**. Виведіть результат у консоль.

Task 03.03. Function Overloading

1. Додайте в першому рядку **app.ts** опцію для ESLint **/* eslint-disable no-redeclare */**. Ця опція необхідна для оголошення кількох сигнатур функцій з однаковими іменами
2. Створіть функцію **getTitles()**, яка може приймати 1 або 2 параметри:
 - а. якщо функція приймає 1 параметр, він може бути або string (author), або boolean (available)
 - б. якщо функція приймає 2 параметри, вони повинні бути id та available.

Функція повинна повертати масив книг за автором, чи за доступністю, чи за id та доступністю.

Для реалізації функції створіть три сигнатури з різними типами параметрів та реалізацію з рест параметром типу any[] або unknown[] або [string | boolean] | [number, boolean].

Функція повинна аналізувати кількість і типи параметрів за допомогою оператора **typeof** і формувати результуючий масив з масиву, отриманого за допомогою функції **getAllBooks()**, аналізуючи властивості: book.author, book.available, book.id.

3. Оголосіть змінну **checkedOutBooks** та викличте функцію **getTitles(false)**. Виведіть результат у консоль.

Task 03.04. Assertion Functions

1. Створіть функцію-ствердження **assertStringValue()**, яка приймає один параметр типу **any**. Функція повинна перевіряти, чи є тип переданого аргументу рядком. Якщо ні, то генерувати виняток **"value should have been a string"**.
2. Створіть функцію **bookTitleTransform()**, яка приймає один параметр – назву книжки (тип параметру **any**). За допомогою **assertStringValue** перевіряє, чи назва книжки дійсно є рядком, і якщо так, то повертає перевертень цього рядка, використовуючи спред оператор і методи масиву **reverse()** і **join()**.
3. Викличте функцію **bookTitleTransform()** двічі і передайте їй рядкове та числове значення.

04. Interfaces

Task 04.01. Defining an Interface

1. Оголосіть інтерфейс **Book**, який включає такі поля:
 - a. `id` - число
 - b. `title` - рядок
 - c. `author` - рядок
 - d. `available` - логічний
 - e. `category` – категорія
2. Внесіть зміни в функцію **getAllBooks()**, вкажіть тип змінної **books** і тип значення, що повертається, використовуючи оголошений вище інтерфейс **Book**. Додайте модифікатор **readonly**. Видаліть тимчасово **id** у книжки та побачите, що з'явиться помилка.
3. Внесіть зміни в функцію **getBookById()**, вкажіть тип **Book['id']** для параметра **id**, а також вкажіть тип значення, що повертається, використовуючи оголошений вище інтерфейс **Book**. Можливо, доведеться додати об'єднання з типом **undefined**, оскільки метод **find**, якщо не знайде елемент, поверне **undefined**.
4. Створіть функцію **printBook()**, яка на вхід приймає книгу та виводить у консоль фразу `book.title + by + book.author`. Використайте інтерфейс **Book** для типу параметра.
5. Оголосіть змінну **myBook** і надайте їй наступний об'єкт

```
{  
  id: 5,  
  title: 'Colors, Backgrounds, and Gradients',  
  author: 'Eric A. Meyer',  
  available: true,  
  category: Category.CSS,  
  year: 2015,  
  copies: 3  
}
```

6. Викличте функцію **printBook()** та передайте їй **myBook**. Жодних помилок при цьому не повинно з'являтися.
7. Додайте до інтерфейсу **Book** властивість **pages: number**. Ви отримаєте помилку у функції **getAllBooks()**. Щоб помилка не виникала, зробіть властивість необов'язковою.
8. Вкажіть явно для змінної **myBook** тип **Book**. Ви знову отримаєте помилку. Видаліть властивості **year, copies**. Додайте властивість **pages: 200**.
9. Додайте в інтерфейс **Book** необов'язкову властивість **markDamaged**, яка є методом. Метод приймає на вхід рядковий параметр **reason** і нічого не повертає. Додайте цей метод до **myBook**. Метод повинен виводити рядок **'Damaged: \${reason}'**. Викличте цей метод та передайте рядок **'missing back cover'**.

Task 04.02. Defining an Interface for Function Types

1. Оголосіть інтерфейс **DamageLogger**, який описуватиме тип функції, яка приймає один рядковий параметр і нічого не повертає.
2. Внесіть зміни до інтерфейсу **Book**: використовуйте оголошений інтерфейс **DamageLogger** для поля **markDamaged**.
3. Оголосіть змінну **logDamage**, використовуючи оголошений раніше інтерфейс **DamageLogger**. Створіть функцію, яка задовольняє цьому інтерфейсу, і надайте її оголошеній змінній. Викличте функцію.

Task 04.03. Extending Interface

1. Оголосіть інтерфейс **Person**, який містить дві рядкові властивості – name і email.
2. Оголосіть інтерфейс **Author** на основі інтерфейсу **Person**, який розширює вказаний інтерфейс числовою властивістю **numBooksPublished**.
3. Оголосіть інтерфейс **Librarian** на основі інтерфейсу **Person**, який розширює цей інтерфейс двома властивостями:
 - a. Рядкова властивість **department**
 - b. Функція **assistCustomer**, яка приймає два рядкові параметри **custName** і **bookTitle** і нічого не повертає.
4. Оголосіть змінну **favoriteAuthor**, використовуючи інтерфейс **Author**, задайте значення у вигляді літерала об'єкта.
5. Оголосіть змінну **favoriteLibrarian**, використовуючи інтерфейс **Librarian**, задайте значення у вигляді літерала об'єкта.

Task 04.04. Optional Chaining

1. Оголосіть змінну **offer** наступного виду:

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

2. Виведіть у консоль значення таких виразів, використовуючи оператор optional chaining (?.)

- a. offer.magazine
- b. offer.magazine.getTitle()
- c. offer.book.getTitle()
- d. offer.book.authors[0]

Task 04.05. keyof Operator

1. Оголосіть тип **BookProperties**, який включає властивості інтерфейсу **Book**, використовуючи **keyof** оператор.
2. Реалізуйте функцію **getProperty()**, яка приймає два параметри:
 - a. книжку
 - b. назву властивості з інтерфейсу Book

і повертає значення цієї властивості з переданого об'єкта, якщо це не функція, для функції повертає її ім'я. Використовуйте тип **any** для значення, що повертається.
3. Викличте цю функцію тричі зі значенням другого параметра: **title**, **markDamaged**, **isbn**.

05. Classes

Task 05.01. Creating and Using Classes

1. Створіть клас **Referenceltem**, який містить:
 - a. Рядкову властивість **title**
 - b. Числову властивість **year**
 - c. Конструктор з двома параметрами: рядковий параметр **newTitle**, числовий параметр **newYear**, який у консоль виводить рядок '**Creating a new Referenceltem...**' та ініціалізує властивості.
 - d. Метод **printItem()** без параметрів, що нічого не повертає. Цей метод повинен виводити рядок '**title was published in year**' в консоль.
2. Оголосіть змінну **ref** та проініціалізуйте її об'єктом **Referenceltem**. Передайте значення для параметрів конструктора. Викличте метод **printItem()**.
3. Закоментуйте конструктор, властивості **title** та **year** та реалізуйте створення властивостей через параметри конструктора (**title - public, year - private**).
4. Створіть приватну ("soft private") рядкову властивість **_publisher**.
 - a. Додайте гетер **publisher**, який перетворює властивість **_publisher** у верхній регістр і повертає його.
 - b. Додайте сеттер **publisher**, який приймає рядковий параметр **newPublisher** і встановлює значення властивості **_publisher** в значення цього параметра.
 - c. Проініціалізуйте властивість **ref.publisher** будь-яким рядковим значенням і виведіть його в консоль. Результат має бути у верхньому регістрі.
5. Створіть приватну ("hard private") числову властивість **id**.
 - a. Внесіть зміни до конструктора для ініціалізації цієї властивості.
 - b. Додайте метод **getID()**, який повинен повертати значення властивості **id**.
 - c. Виведіть об'єкт у консоль.
 - d. Викличте метод **getID()**.
6. Створіть статичну рядкову властивість **department** і проініціалізуйте її будь-яким значенням за замовчуванням. Внесіть зміни до методу **printItem()** – виводьте значення цієї статичної властивості у консоль.

Task 05.02. Extending Classes

1. Створіть клас **Encyclopedia** як спадкоємця класу **Referenceltem**. Додайте одну додаткову числову публічну властивість **edition**. Використайте параметри конструктора.
2. Оголосіть змінну **refBook** та створіть об'єкт **Encyclopedia**. Викличте метод **printItem()**;
3. Перевизначте метод **printItem()**. Додайте ключове слово **override**. Нехай він робить те, що робив та додатково виводить рядок у консоль «**Edition: edition (year)**». Ви отримаєте помилку, що властивість **year** недоступна. Щоб властивість стала доступна, змініть модифікатор доступу в класі **Referenceltem** з **private** на **protected**.

Task 05.03. Creating Abstract Classes

1. Внесіть зміни до класу **ReferenceItem** – зробіть його абстрактним.
2. Додайте абстрактний метод **printCitation()**, який не приймає параметрів і не повертає значення. Цей метод має бути без реалізації. Після цього Ви отримаєте помилку в класі **Encyclopedia**, яка повідомлятиме, що не реалізовано абстрактний метод.
3. Додайте реалізацію методу **printCitation** до класу **Encyclopedia**. Метод повинен виводити в консоль рядок "**title - year**".
4. Оголосіть змінну **refBook** та проініціалізуйте її об'єктом **Encyclopedia**. Викличте метод **printCitation()**;

Task 05.04. Interfaces for Class Types

1. Створіть клас **UniversityLibrarian**, який реалізує інтерфейс **Librarian** та реалізуйте всі необхідні властивості. Метод **assistCustomer** повинен виводити в консоль рядок ``${this.name} is assisting ${custName} with book ${bookTitle}``.
2. Оголосіть змінну **favoriteLibrarian** за допомогою інтерфейсу **Librarian** і проініціалізуйте її за допомогою об'єкта, створеного класом **UniversityLibrarian()**. Жодних помилок при цьому не повинно виникати. Проініціалізуйте властивість **name** та викличте метод **assistCustomer()**.

Task 05.05. Intersection and Union Types

1. Створіть тип **PersonBook**. Використовуйте для цього інтерфейси **Person**, **Book** та перетин типів.
2. Оголосіть змінну з типом **PersonBook**, проініціалізуйте її літералом, виведіть її в консоль.
3. Створіть тип **BookOrUndefined**. Використовуйте для цього об'єднання інтерфейсу **Book** та **undefined**.
4. Замініть тип значення, що повертається у функції **getBookByID()** на **BookOrUndefined**.
5. Створіть функцію **setDefaultConfig()**, яка приймає об'єкт **options**. Тип для об'єкта **TOptions** опишіть інтерфейсом з необов'язковими числовими властивостями **duration** і **speed**. Функція повинна встановлювати значення властивостей за замовчуванням та деякі значення, якщо вони не задані, використовуючи логічний оператор налогового присвоєння та повертати об'єкт.

06. Modules and Namespaces

Task 06.01. Using Namespaces

1. Створіть папку для нового проекту **NamespaceDemo**
2. Створіть файл **utility-functions.ts**
3. Створіть простір імен **Utility**
4. Створіть та експортуйте вкладений простір імен **Fees**
5. Створіть та експортуйте функцію **calculateLateFee()** у вкладеному просторі імен, яка приймає числовий параметр **daysLate** та повертає **fee**, обчислене як $\text{daysLate} * 0.25$;
6. Створіть та експортуйте функцію **maxBooksAllowed()** у просторі імен **Utility**, яка приймає один числовий параметр **age**. Якщо $\text{age} < 12$, то повертає 3 або 10.
7. Створіть функцію **privateFunc()**, яка виводить у консоль повідомлення «**This is a private function**»
8. Створіть файл **app.ts**. Додайте посилання на файл **utility-functions.ts**
9. Напишіть фрагмент коду, який використовує функції із простору імен.
10. Використайте ключове слово **import** та оголосіть аліас **util** для вкладеного простору імен.
import util = Utility.Fees;
11. Запустіть компілятор та скомпілюйте лише **tsc app.ts --target ES5**. Створіть **index.html**
Скористайтесь наступним фрагментом HTML:

```
<html>
  <head></head>
  <body>
    <script src="utility-functions.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```

12. Запустіть компілятор ще раз і вкажіть опцію **--outFile bundle.js**
13. Підключіть отриманий файл до **index.html**

Task 06.02. Export and Import

1. Створіть файл **enums.ts**, перенесіть до нього **enum Category**. Додайте експорт в кінці файлу.
2. Створіть файл **interfaces.ts** та
 - a. перенесіть до нього інтерфейси: **Book, DamageLogger, Person, Author, Librarian**
 - b. додайте імпорт **Category**
 - c. додайте експорт інтерфейсів **Book, DamageLogger, Person, Author, Librarian, TOptions** в кінці файлу. Експортуйте **DamageLogger** під назвою **Logger**
3. Створіть новий файл **classes.ts** та перенесіть до нього класи: **UniversityLibrarian, Referenceltem**.
 - a. Додайте імпорт інтерфейсів як цілого модуля з ім'ям **Interfaces**
 - b. Змініть опис класу **UniversityLibrarian**, щоб він реалізовував інтерфейс **Interfaces.Librarian**
 - c. Додайте експорт в кінці файлу та екпортуйте обидва класи.
4. Створіть файл **types.ts** і перенесіть у нього типи: **BookProperties, PersonBook, BookOrUndefined**.
 - a. Додайте імпорт інтерфейсів **Book** та **Person**
 - b. Екпортуйте типи із модуля.
5. Створіть файл **functions.ts** та перенесіть усі функції.
 - a. Додайте імпорт інтерфейсу **Book, enum Category**, типів **BookProperties, BookOrUndefined**
 - b. Додайте експорт всіх функцій (не обов'язково)
6. Внесіть зміни до файлу **app.ts**
 - a. Додайте імпорт категорій, інтерфейсів **Book, Logger, Author, Librarian**, класів **UniversityLibrarian, Referenceltem**, типу **PersonBook** та всіх функцій.
 - b. Змініть тип змінної **logDamage** на **Logger** (Task 04.02)

Task 06.03. Default Export

1. Створіть файл **encyclopedia.ts** та перемістіть до нього клас **Encyclopedia**. Додайте імпорт **Referenceltem**. Додайте експорт за замовчуванням.
2. Імпортуйте цей клас у **app.ts** як **RefBook**
3. Внесіть зміни до коду завдання **Task 05.02**.
4. / Автор: Yevhen_Zakharevych@epam.com /. Створіть функцію-ствердження умови **assertRefBookInstance** в модулі **functions.ts** Функція повинна приймати **condition: any** та повертати тип **asserts condition**. Якщо умова не виконується, функція повинна генерувати виняток «**It is not an instance of RefBook**» .
5. Створіть та експортуйте функцію **printRefBook(data: any): void**, яка використовує функцію **assertRefBookInstance** та викликає метод **printItem()** у екземпляра **RefBook**. Умову перевірки задайте за допомогою оператора **instanceof**
6. Імпортуйте функцію **printRefBook** в **app.ts** та викличте для екземпляра класу **RefBook**.
7. Створіть екземпляр класу **UniversityLibrarian** та знову викличте для нього функцію **printRefBook**

Task 06.04. Re-Export

1. Створіть папку **classes** і перемістіть файл **encyclopedia.ts** до неї.
2. Рознесіть класи **UniversityLibrarian** і **ReferenceItem** по різних файлах і перемістіть в папку **classes**.
3. Видаліть файл **classes.ts**
4. Створіть файл **classes/index.ts** і додайте до нього реекспорт класів **Encyclopedia**, **ReferenceItem**, використовуючи конструкцію **export ***, **export { default as ...}**, а також додайте реекспорт класу **UniversityLibrarian**, використовуючи конструкцію **export * as UL**.
5. Виправте імпорти у файлі **app.ts**
6. Виправте створення екземпляра класу **UniversityLibrarian** у завданні **Task 05.04.** та **Task 06.03**

Task 06.05. Dynamic Import Expression

1. Створіть у папці **classes** файл **reader.ts** та реалізуйте клас **Reader**, який містить такі властивості:
 - a. `name: string;`
 - b. `books: Book[] = [];`
 - c. `take(book: Book): void` - метод додає книжку до масиву книжок.
2. Внесіть зміни до файлу **classes/index.ts**, додайте новий модуль.
3. Реалізуйте вираз динамічного імпорту за допомогою виразу **top level await/Promise** для завантаження всього з шляху **'./classes'** як модуля. Завантаження реалізувати за умови, якщо деяка змінна набуває значення **true**.
4. Додайте до `webpack.config.js` об'єкт

```
experiments: {  
  topLevelAwait: true  
}
```

5. Створіть екземпляр класу **Reader**. Виведіть його в консоль.

Task 06.06. Type-Only Imports and Exports

1. Створіть у папці **classes** файл **library.ts** та реалізуйте клас **Library**, який містить такі властивості:
 - a. `Id: number`
 - b. `name: string`
 - c. `address: string`
2. Внесіть зміни до файлу **classes/index.ts**. Експортуйте тип **Library**. Використовуйте конструкцію **export type {...}**
3. Імпортуйте **Library** в **app.ts**. Оголосіть змінну за допомогою **Library**.
4. Створіть екземпляр класу **Library**. Ви повинні отримати помилку. Закоментуйте рядок.
5. Оголосіть змінну, вкажіть тип **Library**. Проініціалізуйте літералом, виведіть у консоль.

07. Generics

Task 07.01. Generic Functions

1. Створіть у файлі **functions.ts** дженерик (загальну) функцію **purge()**, яка приймає один параметр – дженерик масив **inventory** та повертає дженерик масив того ж типу, що містить елементи початкового масиву без двох перших елементів. Експоруйте цю функцію.
2. Імпоруйте цю функцію у програму.
3. Додайте категорію **Software** у файл **enums.ts**.
4. Оголосіть змінну **inventory**, що містить наступний масив 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. Викличте функцію **purge()** та передайте їй ці дані. Виведіть результат у консоль.
6. Викличте цю функцію, але з числовим масивом і знову виведіть результат у консоль.

Task 07.02. Generic Interfaces and Classes

1. Створіть інтерфейс **Magazine**, який містить дві рядкові властивості, **title**, **publisher** та додайте його у файл **interfaces.ts**. Експортуйте цей інтерфейс.
2. Створіть файл **classes/shelf.ts** і, використовуючи експорт за замовчуванням, реалізуйте дженерик клас **Shelf**:
 - а. додайте приватну властивість **items**, яка є масивом елементів типу **T**.
 - б. додайте метод **add()**, який приймає один параметр **item** типу **T** і додає його в масив. Нічого не повертає.
 - в. додайте метод **getFirst()**, який нічого не приймає, і повертає перший елемент із **items**.
3. Додайте реекспорт у файл **classes/index.ts**
4. Імпортуйте цей клас і інтерфейс **Magazine** в **app.ts**.
5. Закоментуйте код, який відноситься до функції **purge()**, крім змінної **inventory**
6. Створіть полку **bookShelf** і збережіть усі книжки з **inventory** на полку. Отримайте першу книжку і виведіть її назву в консоль.
7. Об'явіть змінну **magazines**, яка містить наступні дані:

```
[
  { title: 'Programming Language Monthly', видавець: 'Code Mags' },
  { title: 'Literary Fiction Quarterly', видавець: 'College Press' },
  { title: 'Five Points', видавець: 'GSU' }
];
```
8. Створіть полку **magazineShelf** і помістіть усі ці журнали на полку. Отримайте перший журнал і виведіть його в консоль.

Task 07.03. Generic Constraints

1. Внесіть зміни в клас **Shelf**:

- a. додайте метод **find()**, який приймає рядковий параметр **title** і повертає перший знайдений елемент на полиці типу **T**.
- b. додайте метод **printTitles()**, який виводить у консоль назву того, що знаходиться на полиці.

Після додавання цих методів ви повинні отримати помилку, оскільки властивість **title** не існує.

2. У файлі **interfaces.ts** створіть інтерфейс **ShelfItem**, який повинен містити всі необхідні властивості, які повинен мати тип **T**, а саме **title**.
3. Додайте дженерик обмеження для класу, розширив тип **T**.
4. Викличте метод **printTitles()** для журналів.
5. Знайдіть журнал **'Five Points'** і виведіть його в консоль.
6. Створіть функцію **getObjectProperty()**. Додайте два параметра типу **TObject**, **TKey**. Додайте обмеження на другий параметр, щоб значення були тільки ключами об'єкта типу **TObject**, використовуючи оператор **keyof**. Для значення, яке повертається, вкажіть тип **TObject[TKey] | string**. Тіло функції аналогічне тілу функції **getProperty()**. Викличте цю функцію.

Task 07.04. Utility Types

1. Оголосіть аліас типу **BookRequiredFields** у файлі **types.ts**, використовуючи інтерфейс **Book** та утиліту **Required**.
2. Оголосіть змінну типу **BookRequiredFields** та надайте їй відповідний об'єкт.
3. Оголосіть аліас типу **UpdatedBook**, використовуючи інтерфейс **Book** та утиліту **Partial**
4. Оголосіть змінну **updatedBook** і надайте їй відповідний об'єкт.
5. Оголосіть аліас типу **AuthorWoEmail**, використовуючи інтерфейс **Author** та утиліту **Omit**.
6. Оголосіть аліас **CreateCustomerFunctionType** для функціонального типу функції **createCustomer**.
7. Оголосіть змінну, використовуючи аліас типу **CreateCustomerFunctionType** і утиліту **Parameters**, викличте функцію **createCustomer**, передавши цю змінну.

Task 07.05. Mapped Types, Utility Types, Conditional Types

1. Оголосіть у файлі **types.ts** аліас **fn** для функціонального типу функції, яка приймає три параметри з типами `string`, `number`, `boolean` і повертає тип `symbol`.
2. Оголосіть аліаси типів **Param1<T>**, **Param2<T>**, **Param3<T>**, які повертають тип першого, другого та третього параметрів функції відповідно.
3. Оголосіть аліаси **P1**, **P2**, **P3** та отримайте типи першого, другого та третього параметрів типу **fn**.

Автор: Olena_Hlukhovska@epam.com

4. Створіть утиліти **RequiredProps<T>** та **OptionalProps<T>** у файлі **types.ts**, які повертають `union` тип **required** та **optional** властивостей об'єкта. Використовуйте **mapped type** для перебору ключів `T` та **conditional type** для трансформації значень ключів типу `T`. Додайте дженерик обмеження для `T` розширивши тип `object` у **RequiredProps** та **OptionalProps**.
5. Оголосіть аліас типу **BookRequiredProps** та **BookOptionalProps**, використовуючи інтерфейс **Book** та утиліти **RequiredProps** та **OptionalProps**. Спробуйте замість **Book** передати примітивний тип.
6. Створіть утиліту **RemoveProps <T extends object, TProps extends keyof T>**, яка видаляє властивості **TProps** з переданого типу `T`.
7. Оголосіть аліас типу **BookRequiredPropsType** та **BookOptionalPropsType**, використовуючи інтерфейс **Book**, аліаси типу **BookRequiredProps** та **BookOptionalProps** та утиліту **RemoveProps**. Спробуйте замість **Book** передати **Author**.

08. Decorators

Task 08.01. Class Decorators (sealed)

1. Створіть файл **decorators.ts**. Створіть декоратор класу **@sealed()**, щоб запобігти додаванню нових властивостей об'єкту класу та прототипу об'єкта. Функція-декоратор повинна приймати один рядковий параметр і нічого не повертати. Перед виконанням функціонала функція має вивести у консоль повідомлення **"Sealing the constructor + параметр"**. Використовуйте метод **Object.seal()**.
2. Застосуйте цей декоратор до класу **UniversityLibrarian**.
3. Створіть екземпляр класу **UniversityLibrarian**. Перевірте повідомлення у консолі.

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

1. Створіть декоратор класу **@logger()**, який змінюватиме конструктор класу.
2. Оголосіть всередині декоратора змінну **newConstructor: Function** та проініціалізуйте її функціональним виразом. Новий конструктор повинен:
 - a. виводити в консоль повідомлення **"Creating new instance"**
 - b. виводити переданий параметр (ім'я класу).
 - c. створювати нову властивість **age** зі значенням 30.
3. Проініціалізуйте прототип нового конструктора об'єктом, створеним на основі прототипу переданого класу, використовуючи **Object.create()** або **Object.setPrototypeOf()**.
4. Додайте новий метод до прототипу нового конструктора **printLibrarian()**, який повинен виводити в консоль рядок **`Librarian name: \${this.name}, Librarian age: \${this.age}`**.
5. Поверніть з декоратора новий конструктор, попередньо перетворивши його на тип **<TFunction>**.
6. Застосуйте цей декоратор до класу **UniversityLibrarian**. Перевірте результат роботи в консолі.
7. Оголосіть змінну **fLibrarian** та створіть екземпляр класу **UniversityLibrarian**. Вкажіть значення Anna для name. Викличте метод **printLibrarian()**.

Task 08.03. Method Decorator (writable)

1. Створіть декоратор методу **@writable()** як фабрику, яка отримує булевий параметр **isWritable**. Декоратор повинен встановлювати властивість дескриптора **writable** у передане значення.
2. Додайте два методи для класу **UniversityLibrarian**:
 - a. **assistFaculty()** – виводить у консоль повідомлення «**Assisting faculty**».
 - b. **teachCommunity()** – виводить у консоль повідомлення «**Teaching community**».
3. Задекоруйте метод **assistFaculty()** як змінний, а метод **teachCommunity()** як незмінний.
4. Спробуйте змінити методи у екземпляра цього класу.

Task 08.04. Method Decorator (timeout)

1. Створіть декоратор методу **@timeout()** як фабрику, яка отримує числовий параметр – кількість мілісекунд. Метод, до якого застосовується декоратор, повинен запускатися через вказану кількість часу і тільки, якщо користувач дав на це згоду за допомогою підтверджуючого вікна (`window.confirm`)
2. Декоратор повинен перевизначати властивість дескриптора `value`. Нова функція повинна використовувати **setTimeout()** та запускати початковий метод через вказану кількість часу. Поверніть з декоратора новий дескриптор.
3. Застосуйте декоратор до методу **printItem()** класу **ReferenceItem**.
4. Створіть екземпляр класу **Encyclopedia** та викличте метод **printItem()**

Task 08.05. Parameter Decorator (logParameter)

1. Створіть декоратор параметра методу - **@logParameter()**, який повинен зберігати індекс параметра, до якого застосовується декоратор у властивість прототипу **`\${methodName}_decor_params_indexes`**. Властивість організувати як масив.
2. Створіть декоратор методу **@logMethod()**. Декоратор повинен перевизначати метод, до якого він застосовується та повертати новий дескриптор.
3. Перевизначений метод повинен отримати доступ до індексів, що знаходяться у властивості **`\${methodName}_decor_params_indexes`** і для кожного параметра виводити його значення у форматі **Method: `\${methodName}`, ParamIndex: `\${ParamIndex}`, ParamValue: `\${ParamValue}`**
4. Задекоруйте метод **assistCustomer()** та всі його параметри відповідними декораторами.
5. Створіть екземпляр класу **UniversityLibrarian**, проініціалізуйте властивість **name**, викличте метод **assistCustomer()**.

Task 08.06. Property Decorator

1. Створіть фабричну функцію декоратора властивості **@format(pref:string = 'Mr./Mrs.')**, яка при застосуванні до властивості форматує його значення – додає префікс **pref**. Фабрична функція повинна повертати функцію з сигнатурою декоратора властивості, всередині якої необхідно викликати функцію **makeProperty(target, propertyName, value => `\${pref} \${value}`, value => value)**;
2. Функція **makeProperty** має такий вигляд:

```
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 {  
            return 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. Додайте функцію **makeProperty** до **decorators.ts**
4. Задекоруйте властивість **name** класу **UniversityLibrarian** декоратором **@format()**
5. Створіть екземпляр класу **UniversityLibrarian**. Встановіть значення для властивості **name**, потім отримайте його та виведіть у консоль.

Task 08.07. Accessor Decorator

1. Створіть декоратор аксесору **@positiveInteger()**, який кидає виняток у випадку, якщо властивості встановлюється значення менше 1 і не ціле.
2. Додайте до класу **Encyclopedia** приватну числову властивість **_copies**, а також геттер і сеттер для цієї властивості, які повертають значення та встановлюють значення відповідно.
3. Задекоруйте геттер або сеттер декоратором **@positiveInteger()**.
4. Створіть екземпляр класу **Encyclopedia**. Спробуйте встановити різні значення **-10, 0, 4.5, 5**

09. Asynchronous Patterns

Task 09.01. Callback Functions

1. У файлі **interfaces.ts** створіть інтерфейс для функції зворотного виклику **LibMgrCallback**, який приймає два параметри:
 - a. **err: Error | null**,
 - b. **titles: string[] | null**і нічого не повертає
2. У файлі **interfaces.ts** створіть дженерик інтерфейс **Callback<T>** з двома властивостями:
 - a. **err: Error | null**,
 - b. **data: T | null**
3. У файлі **functions.ts** створіть функцію **getBooksByCategory()**, яка приймає два параметри:
 - a. **category: Category**
 - b. **callback** – тип, раніше створений інтерфейсі нічого не повертає
4. Функція повинна використовувати **setTimeout()** та через 2с виконати наступний код:
 - a. У розділі **try**: використовувати функцію **getBookTitlesByCategory()** для отримання заголовків книг за категорією
 - b. Якщо знайшли книги, то викликати функцію зворотного виклику та передати два параметри: **null** та знайдені книги
 - c. Якщо не знайшли книг, то кинути виняток **throw new Error('No books found.');**
 - d. У секції **catch**: викликати функцію зворотного виклику та передати два параметри **error** і **null**.
5. Створіть функцію **logCategorySearch()**, яка має сигнатуру, описану в інтерфейсі **LibMgrCallback** або **Callback**. Якщо прийшов об'єкт помилки, то вивести властивість **err.message**, інакше вивести назви книг.
6. Викличте функцію **getBooksByCategory()** та передайте їй необхідні аргументи. Додайте виведення повідомлень у консоль перед та після виклику цієї функції. Використовуйте **Category.JavaScript** та **Category.Software** як значення першого параметра.

Task 09.02. Promises

1. Створіть функцію **getBooksByCategoryPromise()**, яка приймає один параметр – **category** та повертає проміс – масив заголовків книг.
2. Використовуйте **new Promise((resolve, reject) => { setTimeout(() => {...}, 2000) });** Додайте код, аналогічний функції **getBooksByCategory()**, тільки тепер використовуйте **resolve()** та **reject()**. Поверніть із функції створений проміс.
3. Викличте функцію **getBooksByCategoryPromise()** та зареєструйте функції зворотного виклику за допомогою методів **then** та **catch**. Додайте виведення повідомлень у консоль перед та після виклику цієї функції. Використовуйте **Category.Javascript** та **Category.Software** як значення параметра.
4. Поверніть кількість знайдених книг із функції, зареєстрованої за допомогою **then()**. Зареєструйте за допомогою іншого методу **then()** функцію, яка повинна вивести в консоль кількість знайдених книг.
5. У файлі **types.ts** створіть аліас типу **Unpromisify<T>**, який повинен повертати тип значення промісу.
6. Отримайте тип значення функції **getBooksByCategoryPromise()**, що повертається, використовуючи **typeof** оператор і утиліту **ReturnType**
7. Застосуйте **Unpromisify<T>** до отриманого типу, який повертає функція **getBooksByCategoryPromise()**

Task 09.03. Async Functions

1. Створіть асинхронну функцію **logSearchResults()** у файлі **funtions.ts**. Функція повинна використовувати функцію **getBooksByCategoryPromise**, отримувати та виводити в консоль кількість знайдених книг.
2. Викличте цю функцію. Вкажіть значення параметра **Category.Javascript**. Додайте вивід у консоль до та після виклику функції. Опрацюйте помилку за допомогою **catch**