TypeScript In-Depth

Practice Document

Contents

02. Types Basics

Завдання 02.01 Базові типи	**Task 02.01. Basic Types**	**Задание 02.01. Базовые типы**
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: • кількість книг у масиві • назву першої доступної книги Запустіть функцію **logFirstAvailable**() 3 Об'явіть **enum Category** для зберігання наступних категорій книг: JavaScript, CSS, HTML, TypeScript, Angular. Додайте категорію до об'єктів у функції **getAllBooks**().	1 Implement a **getAllBooks**() function, which returns a collection of books. Declare this collection inside a function. [{ 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 Implement a **logFirstAvailable**() function that takes an array of books as a parameter and prints to the console: • number of books in the array • title of the first available book Run the **logFirstAvailable**() function. 3 Declare an **enum Category** to store the following book categories: JavaScript, CSS, HTML, TypeScript, Angular. Add a category to the objects in the **getAllBooks**() function.	1 Реализуйте функцию **getAllBooks()**, которая возвращает коллекцию книжек. Объявите эту коллекцию внутри функции. [{ 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: • количество книг в массиве • название первой доступной книги Запустите функцию **logFirstAvailable().** 3 Объявите **enum Category** для хранения следующих категорий книг: JavaScript, CSS, HTML, TypeScript, Angular. Добавьте категорию к объектам в функции **getAllBooks().**

4 Реалізуйте функцію **getBookTitlesByCategory**(), яка на вхід повинна отримувати категорію та повертати масив найменувань книг, що належать зазначеній категорії. 5 Реалізуйте функцію **logBookTitles**(), яка повинна приймати масив рядків та виводити його в консоль. Викличте функції **getBookTitlesByCategory**() та **logBookTitles**(). 6 Реалізуйте функцію **getBookAuthorByIndex**(), яка повинна приймати **index** книжки у масиві та повертати пару: назву книжки + автор. Використовуйте **tuple** для типу, що повертається. Викличте цю функцію. Внесіть зміни до типу, що повертається функцією **getBookAuthorByIndex**() – додайте мітки: title, author для типу tuple. 7 Реалізуйте функцію **calcTotalPages**(), яка повинна підраховувати кількість сторінок книг у трьох бібліотеках міста, використовуючи такі дані: [{ lib: 'libName1', books: 1_000_000_000, avgPagesPerBook: 250 },	4 Implement a **getBookTitlesByCategory**() function, which should take a category as input and return an array of book titles that belong to the specified category. 5 Implement a **logBookTitles**() function that should take an array of strings and print it to the console. Call the **getBookTitlesByCategory**() and **logBookTitles**() functions. 6 Implement a **getBookAuthorByIndex**() function, which should take the index of the books in the array and return the pair: book title + author. Use tuple for the return type. Call this function. Make changes to the type returned by the **getBookAuthorByIndex**() function - add labels: title, author for the tuple type. 7 Implement a **calcTotalPages**() function that should count the number of book pages in a three libraries in a city using the following data: [{ lib: 'libName1', books: 1_000_000_000, avgPagesPerBook: 250 }, { lib: 'libName2', books: 5_000_000_000, avgPagesPerBook: 300 },	4 Реализуйте функцию **getBookTitlesByCategory()**, которая на вход должна получать категорию и возвращать массив наименований книг, которые принадлежат указанной категории. 5 Реализуйте функцию **logBookTitles(),** которая должна принимать массив строк и выводить его в консоль. Вызовите функции **getBookTitlesByCategory()** и **logBookTitles().** 6 Реализуйте функцию **getBookAuthorByIndex(),** которая должна принимать index книжки в массиве и возвращать пару: название книжки + автор. Используйте tuple для возвращаемого типа. Вызовите данную функцию. Внесите изменения в тип возвращаемый функцией **getBookAuthorByIndex()** – добавьте лейблы: title, author для типа tuple. 7 Реализуйте функцию **calcTotalPages()**, которая должна подсчитывать количество страниц книг в трех библиотеках города, используя следующие данные: [{ 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.	{ lib: 'libName3', books: 3_000_000_000, avgPagesPerBook: 280 }]; For calculations, use the bigint type.	{ lib: 'libName2', books: 5_000_000_000, avgPagesPerBook: 300 }, { lib: 'libName3', books: 3_000_000_000, avgPagesPerBook: 280 }]; Для подсчетов используйте тип bigint.

Завдання 02.02 Приведення до константи	**Task 02.02. Const Assertions**	**Задание 02.02. Приведение к константе**
1 Додайте **const assertions** (<const>) для масиву книг та масиву, який містить інформацію про сторінки книг у бібліотеках міста. 2 Додайте модифікатор **readonly** для параметра функції **logFirstAvailable**()	1 Add **const assertions** (<const>) for an array of books and an array that contains information about book pages in a city libraries. 2 Add the **readonly** modifier to the logFirstAvailable() function parameter	1 Добавьте **const assertions** (<const>) для массива книг и массива, который содержит информацию о страницах книг в библиотеках города. 2 Добавьте модификатор **readonly** для параметра функции **logFirstAvailable**()

03. Functions

Завдання 03.01. Функціональний тип	Task 03.01. Functional Type	**Задание 03.01. Функциональный тип**
1 Створіть функцію **createCustomerID**(), яка приймає ім'я клієнта (name: string) та його ідентифікатор (id: number) та повертає конкатенацію цих значень у вигляді рядка.	1 Create a **createCustomerID**() function that takes a customer name (name: string) and an ID (id: number) and returns the concatenation of these values as a string.	1 Создайте функцию **createCustomerID()**, которая принимает имя клиента (name: string) и его идентификатор (id: number) и возвращает конкатенацию этих значений в виде строки.
2 Об’явіть змінну **myID** рядкового типу та викличте функцію зі значеннями Ann, 10. Отримане значення виведіть у консоль.	2 Declare a string variable **myID** and call the function with the values Ann, 10. Print a result to the console.	2 Объявите переменную **myID** строчного типа и вызовите функцию с значениями Ann, 10. Полученное значение выведите в консоль.
3 Об’явіть змінну **idGenerator** і вкажіть тип функції **createCustomerID**(). Надайте цій змінній функціональний вираз, використовуючи стрілочну функцію. Тіло подібне до функції **createCustomerID**().	3 Declare an **idGenerator** variable and set the type of the **createCustomerID**() function. Assign a function expression to this variable using an arrow function. The body is similar to the **createCustomerID**() function.	3 Объявите переменную **idGenerator** и задайте тип функции **createCustomerID()**. Присвойте этой переменной функциональное выражение, используя стрелочную функцию. Тело аналогично функции **createCustomerID()**.
4 Надайте змінній **idGenerator** функцію **createCustomerID**() та викличте її. Отримане значення виведіть у консоль.	4 Assign **createCustomerID**() function to the **idGenerator** variable and call it. Print a result to the console.	4 Присвойте переменной **idGenerator** функцию **createCustomerID()** и вызовите ее. Полученное значение выведите в консоль.

Завдання 03.02. Необов'язкові, значення за замовчуванням та рест параметри	**Task 03.02. Optional, Default and Rest Parameters**	**Задание 03.02. Необязательные, значение поумолчанию и рест параметры**
1 Створіть функцію **createCustomer**(), яка приймає три параметри: • name: string – обов'язковий • age: number – необов'язковий • 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.	1 Create a **createCustomer**() function that takes three parameters: • name: string - required • age: number - optional • city: string - optional The function should output the client's name to the console, and if the age is given, it should additionally output the age to the console. If a city is given, then it should additionally output the city to the console. Call this function with one, two, and three arguments.	1 Создайте функцию **createCustomer()**, которая принимает три параметра: • name: string – обязательный • age: number – необязательный • city: string – необязательный Функция должна выводить имя клиента в лог, 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 аргументами.
2 Внесіть зміни до функції **getBookTitlesByCategory**() – додайте для параметра значення за замовчуванням **Category.JavaScript**. Викличте цю функцію без аргумента.	2 Modify the **getBookTitlesByCategory**() function - add a default value of **Category.JavaScript** for the parameter. Call this function without an argument.	2 Внесите изменения в функцию **getBookTitlesByCategory()** – добавьте для параметра значение по умолчанию **Category.JavaScript**. Вызовите эту функцию без аргумента.
3 Внесіть зміни до функції **logFirstAvailable**() – додайте для параметра значення за замовчуванням – виклик функції **getAllBooks**(). Викличте цю функцію без аргументів.	3 Make changes to the **logFirstAvailable**() function - add a default value for the parameter - a call to the **getAllBooks**() function. Call this function without an argument.	3 Внесите изменения в функцию **logFirstAvailable() –** добавьте для параметра значение по умолчанию – вызов функции **getAllBooks()**. Вызовите эту функцию без аргумента.
4 Створіть функцію **getBookByID**(), яка приймає id книжки та повертає книжку. Використовуйте функцію **getAllBooks**(), метод масиву **find**() та	4 Create a **getBookByID**() function that takes the id of a book and returns the book. Use the **getAllBooks**() function, the **find**() array method,	4 Создайте функцию **getBookByID()**, которая принимает **id** книжки и возвращает книжку. Используйте функцию **getAllBooks**(), метод

стрілочну функцію. Викличте функцію та передайте їй 1. 5 Створіть функцію **checkoutBooks**(), яка приймає два параметри: • customer: string • 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.	and the arrow function. Call the function and pass 1. 5 Create a **checkoutBooks**() function that takes two parameters: • customer: string • bookIDs: number[] – variable value of book identifiers (rest parameter) The function should check the availability of each book given by the identifier and return an array of titles (title) of books that are available. (book.available = true). Use the **getBookById**() function. The function should also output the name of the specified client to a console. 6 Declare a variable **myBooks** and store the result of calling the **checkoutBooks**('Ann', 1, 2, 4) function into it. Print the result to the console.	массива **find**() и стрелочную функцию. Вызовите функцию и передайте 1. 5 Создайте функцию **checkoutBooks()**, которая принимает два параметра: • customer: string • bookIDs: number[] – переменное значение идентификаторов книжек (рест параметр) 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.

Завдання 03.03. Перевантаження функцій

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 повинна приймати 1 або 2 аргументи:

- якщо функція приймає 1 аргумент, то він повинен бути або string (author), або boolean (available)
- якщо функція приймає 2 аргументи, то вони повинні бути number (id) та boolean (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**.

Task 03.03. Function Overloading

1
Add an option for ESLint **/* eslint-disable no-redeclare */** in the first line of **app.ts**. This option is required to declare multiple function signatures with the same name.

2
Create a **getTitles()** function that should take 1 or 2 arguments:

- if the function takes 1 argument, then it must be either string (author) or boolean (available)
- if the function takes 2 arguments, then they must be number (id) and boolean (available).

The function should return an array of books by author, or by availability, or by id and availability.

To implement the function, create three signatures with different types of parameters and an implementation with a rest parameter of type any[] or unknown[] or [string | boolean] | [number, boolean].

The function should analyze the number and types of parameters using the **typeof** operator and form the resulting array from the array obtained using the **getAllBooks**() function, analyzing the **book.author, book.available, book.id** properties.

Задание 03.03. Перегрузка функций

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 должна принимать 1 или 2 аргумента:

- если функция принимает 1 аргумент, то он должен быть либо string (author), либо boolean (available)
- если функция принимает 2 аргумента, то они должны быть number (id) и boolean (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 рест параметром типа any[] или unknown[] или [string | boolean] | [number, boolean].

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

3 Оголосіть змінну **checkedOutBooks** та викличте функцію **getTitles**(false). Виведіть результат у консоль.	3 Declare a **checkedOutBooks** variable and call the **getTitles(false)** function. Print the result to the console.	3 Объявите переменную **checkedOutBooks** и вызовите функцию **getTitles(false)**. Выведите результат в консоль.

Завдання 03.04. Функції-стрердження	**Task 03.04. Assertion Functions**	**Задание 03.04. Функции-утверждения**
1 Створіть функцію-ствердження **assertStringValue**(), яка повинна приймати один параметр типу **any**. Функція повинна перевіряти, чи є тип переданого аргументу рядком. Якщо ні, то генерувати виняток **"value should have been a string"**.	1 Create an **assertStringValue()** assertion function that should take one parameter of type **any**. The function should check if the type of the passed argument is a string. If not, then throw a **"value should have been a string"** exception.	1 Создайте функцию-утверждение **assertStringValue**(), которая должна принимать 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 Створіть функцію **bookTitleTransform**(), яка повинна приймати один параметр **title** – назву книжки (тип параметру **any**). За допомогою функції **assertStringValue()** повинна перевіряти, чи назва книжки дійсно є рядком, і якщо так, то повинна повертати перевертень цього рядка, використовуючи спред оператор і методи масиву **reverse**() і **join**().	2 Create a function **bookTitleTransform**() that should take one parameter **title** - the title of the book (parameter type **any**). With **assertStringValue()** it should check if the title of the book is actually a string, and if it is, it should return the reverse of that string using the spread operator and the **reverse**() and **join**() array methods.	2 Создайте функцию **bookTitleTransform**(), которая должна принимать один параметр **title** - название книжки (тип параметра **any**). С помощью **assertStringValue**() должна проверять, действительно ли название книжки является строкой, и если да, то должна возвращать перевертишь этой строки, используя спред оператор и методы массива **reverse**() и **join**().
3 Викличте функцію **bookTitleTransform**() двічі і передайте їй рядкове та числове значення.	3 Call the **bookTitleTransform**() function twice and pass it a string value and a number value.	3 Вызовите функцию **bookTitleTransform**() дважды и передайте ей строчное и числовое значение.

04. Interfaces

Завдання 04.01. Об’явлення інтерфейсу	**Task 04.01. Defining an Interface**	**Задание 04.01. Объявление интерфейса**
1 Об’явіть інтерфейс **Book**, який включає такі поля: • id - число • title - рядок • author - рядок • available - логічний • category – категорія	1 Declare a **Book** interface that includes the following fields: • id - number • title - string • author - string • available - boolean • category – category	1 Объявите интерфейс **Book**, который включает следующие поля: • id - число • title - строка • author - строка • available - логический • category – категория
2 Внесіть зміни в функцію **getAllBooks**(), вкажіть тип для змінної **books** і тип для значення, що повертається, використовуючи інтерфейс **Book**. Додайте модифікатор **readonly**. Видаліть тимчасово **id** у книжки. Ви побачите, що з'явиться помилка.	2 Modify the **getAllBooks()** function to specify the type for the **books** variable and the type for the return value using the **Book** interface. Add the **readonly** modifier. Delete temporarily **id** from the book. You will see an error.	2 Внесите изменения в функцию **getAllBooks()**, укажите тип для переменной **books** и тип для возвращаемого значения, используя интерфейс **Book**. Добавьте модификатор **readonly**. Удалите временно id у книжки. Вы увидите, что появится ошибка.
3 Внесіть зміни в функцію **getBookByID**(), вкажіть тип **Book['id']** для параметра **id**, а також вкажіть тип для значення, що повертається, використовуючи інтерфейс **Book**. Можливо, доведеться додати об'єднання з типом **undefined**, оскільки метод **find**, якщо не знайде елемент, поверне **undefined**.	3 Modify the **getBookByID**() function, specify the type **Book['id']** for the **id** parameter, and specify the type for the return value using the **Book** interface. It may be necessary to add a union with type **undefined**, since the find method will return **undefined** if it does not find an element.	3 Внесите изменения в функцию **getBookByID()**, укажите тип **Book[‘id’]** для параметра **id**, а также укажите тип для возвращаемого значения, используя интерфейс **Book**. Возможно, понадобиться добавить объединение с типом **undefined**, поскольку метод **find**, если не найдет элемент, вернет **undefined**.
4 Створіть функцію **printBook**(), яка повинна приймати один параметр - книгу та виводити у консоль фразу **book.title + by + book.author**.	4 Create a **printBook**() function that should take one parameter, a book, and print the phrase	4 Создайте функцию **printBook()**, которая должна принимать один параметр - книгу и выводить в консоль фразу **book.title + by + book.author**.

Використайте інтерфейс **Book** для типу параметра.
 	
5
Об’явіть змінну **myBook** і присвойте 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**.
	book.title + by + book.author to the console. For the parameter type, use the **Book** interface.
	
	5
	Declare a variable **myBook** and assign the following object to it
	{
	id: 5
	title: 'Colors, Backgrounds, and Gradients',
	author: 'Eric A. Meyer',
	available: true
	category:Category.CSS,
	year: 2015
	copies: 3
	}
	
	6
	Call the **printBook**() function and pass it **myBook**. No errors should appear.
	
	7
	Add the **pages: number** property to the **Book** interface. You will get an error in the **getAllBooks()** function. To prevent an error, make the property optional.
	
	8
	Set the type **Book** explicitly for the variable **myBook**. You will get an error again. Delete the properties **year, copies**. Add the **pages: 200** property.
		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 необязательным.
		
		8
		Укажите явно для переменной **myBook** тип **Book**. Вы снова получите ошибку. Удалите свойства **year, copies**. Добавьте свойство **pages: 200**.

9 Додайте в інтерфейс **Book** необов'язкову властивість **markDamaged**, яка є методом. Метод повинен приймати рядковий параметр **reason** і нічого не повертати. Додайте цей метод до **myBook**. Метод повинен виводити рядок **`Damaged: ${reason}`**. Викличте цей метод та передайте рядок **'missing back cover'**.	9 Add an optional **markDamaged** property to the **Book** interface, which is a method. The method should take a string parameter **reason** and return nothing. Add this method to the **myBook** object. The method should output the string **`Damaged: ${reason}`**. Call this method and pass the string **'missing back cover'**	9 Добавьте в интерфейс **Book** необязательное свойство **markDamaged**, которое является методом. Метод должен принимать строчный параметр **reason** и ничего не возвращать. Добавьте этот метод в объект **myBook**. Метод должен выводить строчку **`Damaged: ${reason}`**. Вызовите этот метод и передайте строку **'missing back cover'**

Завдання 04.02. Об’явлення інтерфейсу для функціонального типу	**Task 04.02. Defining an Interface for Function Types**	**Задание 04.02. Объявление интерфейса для функционального типа**
1 Об’явіть інтерфейс **DamageLogger**, який описуватиме тип функції, яка повинна приймати один рядковий параметр і нічого не повертати.	1 Declare the **DamageLogger** interface, which will describe the type for the function, which should take one string parameter and return nothing.	1 Объявите интерфейс **DamageLogger**, который будет описывать тип для функции, которая должна принимать один строчный параметр и ничего не возвращать.
2 Внесіть зміни до інтерфейсу **Book**: використайте інтерфейс **DamageLogger** для поля **markDamaged**.	2 Make changes to the **Book** interface: use the **DamageLogger** interface for the **markDamaged** field.	2 Внесите изменения в интерфейс **Book**: используйте интерфейс **DamageLogger** для поля **markDamaged**.
3 Об’явіть змінну **logDamage**, використовуючи інтерфейс **DamageLogger**. Створіть функцію, яка задовольняє цьому інтерфейсу, і присвойте її змінній **logDamage**. Викличте функцію.	3 Declare the **logDamage** variable using the **DamageLogger** interface. Create a function that satisfies this interface, assign it to the **logDamage** variable. Call the function.	3 Объявите переменную **logDamage,** используя интерфейс **DamageLogger**. Создайте функцию, которая удовлетворяет этому интерфейсу, присвойте ее переменной **logDamage**. Вызовите функцию.

Завдання 04.03. Розширення інтерфейсів	**Task 04.03 Extending Interfaces**	**Задание 04.03. Расширение интерфейсов**
1 Об’явіть інтерфейс **Person**, який містить дві рядкові властивості – **name** і **email**.	1 Declare a **Person** interface that contains two string properties, **name** and **email**.	1 Объявите интерфейс **Person**, который содержит два строчных свойства – **name** и **email**.
2 Об’явіть інтерфейс **Author** на основі інтерфейсу **Person**, який розширює вказаний інтерфейс числовою властивістю **numBooksPublished**.	2 Declare an **Author** interface based on the **Person** interface that extends the specified interface with the **numBooksPublished** numeric property.	2 Объявите интерфейс **Author** на основе интерфейса **Person**, который расширяет указанный интерфейс числовым свойством **numBooksPublished**.
3 Об’явіть інтерфейс **Librarian** на основі інтерфейсу **Person**, який розширює цей інтерфейс двома властивостями: • Рядкова властивість **department** • Функція **assistCustomer**, яка повинна приймати два рядкові параметри **custName** і **bookTitle** і нічого не повертати.	3 Declare a **Librarian** interface based on the **Person** interface that extends the specified interface with two properties: • String property **department** • The **assistCustomer** function, which should take two string parameters **custName** and **bookTitle** and return nothing.	3 Объявите интерфейс **Librarian** на основе интерфейса **Person**, который расширяет указанный интерфейс двумя свойствами: • Строчное свойство **department** • Функция **assistCustomer**, которая должна принимать два строчных параметра **custName** и **bookTitle** и ничего не возвращать.
4 Об’явіть змінну **favoriteAuthor**, використовуючи інтерфейс **Author**, задайте значення у вигляді літерала об'єкта.	4 Declare a **favoriteAuthor** variable using the **Author** interface, set the value as an object literal.	4 Объявите переменную **favoriteAuthor** используя интерфейс **Author**, задайте значение в виде литерала объекта.
5 Об’явіть змінну **favoriteLibrarian**, використовуючи інтерфейс **Librarian**, задайте значення у вигляді літерала об'єкта.	5 Declare a **favoriteLibrarian** variable using the **Librarian** interface, set the value as an object literal.	5 Объявите переменную **favoriteLibrarian** используя интерфейс **Librarian**, задайте значение в виде литерала объекта.

Завдання 04.04. Необов’язковий ланцюжок	**Task 04.04. Optional Chaining**	**Задание 04.04. Необязательная цепочка**

1
Об’явіть змінну **offer** наступного виду:
 	
`const offer: any = {`
`  book: {`
`  title: 'Essential TypeScript',`
`  },`
`};`
 	
2
Виведіть у консоль значення таких виразів, використовуючи оператор (?.)
• offer.magazine
• offer.magazine.getTitle()
• offer.book.getTitle()
• offer.book.authors[0]
• offer.book.authors[0].name
	1
	Declare an **offer** variable like this:
	
	`const offer: any = {`
	`  book: {`
	`  title: 'Essential TypeScript',`
	`  },`
	`};`
	
	2
	Print the value of the following expressions to the console using the operator (?.)
	• offer.magazine
	• offer.magazine.getTitle()
	• offer.book.getTitle()
	• offer.book.authors[0]
	• offer.book.authors[0].name
		1
		Объявите переменную **offer** следующего вида:
		
		`const offer: any = {`
		`  book: {`
		`  title: 'Essential TypeScript',`
		`  },`
		`};`
		
		2
		Выведите в консоль значение следующих выражений, используя оператор (?.)
		• offer.magazine
		• offer.magazine.getTitle()
		• offer.book.getTitle()
		• offer.book.authors[0]
		• offer.book.authors[0].name

Завдання 04.05. keyof оператор	**Task 04.05. keyof operator**	**Задание 04.05. keyof оператор**
1 Об’явіть тип **BookProperties**, який має бути об’єднанням рядкових літеральних типів властивостей інтерфейсу **Book**, використовуючи **keyof** оператор. 2 Створіть функцію **getProperty**(), яка повинна приймати два параметри: • книжку • назву властивості з інтерфейсу **Book** і повертати значення цієї властивості з переданого об'єкта, якщо це не функція, для функції повертати її ім'я. Використайте тип **any** для значення, що повертається. 3 Викличте функцію **getProperty()** тричі зі значенням другого аргумента: **title, markDamaged, isbn**.	1 Declare a **BookProperties** type, which must be the union of the string literal types of properties of the **Book** interface, using the **keyof** operator. 2 Create a **getProperty()** function that should take two parameters: • book • property name from the **Book** interface and return the value of that property from the passed object, if it's not a function, for the function it should return its name. Use the **any** type for the return value. 3 Call the **getProperty()** function three times with the value for the second argument: **title, markDamaged, isbn**.	1 Объявите тип **BookProperties**, который должен быть объединением строчных литеральных типов свойств интерфейса **Book**, используя **keyof** оператор. 2 Создайте функцию **getProperty**(), которая должна принимать два параметра: • книжку • название свойства из интерфейса **Book** и возвращать значение этого свойства из переданного объекта, если это не функция, для функции возвращать ее имя. Используйте тип **any** для возвращаемого значения. 3 Вызовите функцию **getProperty()** три раза со значением для второго аргумента: **title**, **markDamaged**, **isbn**.

05. Classes

Завдання 05.01. Створення та використання класів	**Task 05.01. Creating and Using Classes**	**Задание 05.01. Создание и использование классов**
1 Створіть клас **ReferenceItem**, який містить: • Рядкову властивість **title** • Числову властивість **year**	1 Create a **ReferenceItem** class that contains: • String property **title** • The numeric property **year**	1 Создайте класс **ReferenceItem**, содержащий: • Строковое свойство **title** • Числовое свойство **year**
2 Додайте конструктор, який повинен приймати два параметри: • рядковий параметр **newTitle**, • числовий параметр **newYear**, виводити у консоль рядок **'Creating a new ReferenceItem...'** та ініціалізувати властивості **title** та **year.**	2 Add a constructor that should take two parameters: • string parameter **newTitle**, • numeric parameter **newYear**, print the line **'Creating a new ReferenceItem...'** to the console and initialize the **title** and **year** properties.	2 Добавьте конструктор, который должен принимать два параметра: • строчный параметр **newTitle**, • числовой параметр **newYear**, выводить в консоль строку **'Creating a new ReferenceItem...'** и инициализировать свойства **title** и **year**.
3 Додайте метод **printItem**(), який повинен нічого не приймати і нічого не повертати. Цей метод повинен виводити рядок **"title was published in year"** в консоль.	3 Add a **printItem**() method that should take nothing and return nothing. This method should print the line **"title was published in year"** to the console.	3 Добавить метод **printItem()**, который должен ничего не принимать и ничего не возвращать. Этот метод должен выводить строку **"title was published in year"** в консоль.
4 Об’явіть змінну **ref** та проініціалізуйте її об'єктом **ReferenceItem**. Передайте значення для параметрів конструктора. Викличте метод **printItem**().	4 Declare a **ref** variable and initialize it with a **ReferenceItem** object. Pass values for constructor parameters. Call the **printItem**() method.	4 Объявите переменную **ref** и проинициализируйте ее объектом **ReferenceItem**. Передайте значения параметрам конструктора. Вызовите метод **printItem**().
5 Закоментуйте конструктор, властивості **title** та **year** та реалізуйте створення властивостей	5 Comment out the constructor, the **title** and **year** properties, and implement the creation of the	5 Закомментируйте конструктор, свойства **title** и **year** и реализуйте создание свойств через

через параметри конструктора **title - public, year - private**. 6 Створіть приватну (“soft private”) рядкову властивість **_publisher**. • Додайте гетер **publisher**, який повинен повертати значення властивості **_publisher** у верхньому регістрі. • Додайте сеттер **publisher**, який повинен приймати рядковий параметр **newPublisher** і встановлює значення властивості **_publisher** в значення цього параметра. • Проініціалізуйте властивість **ref.publisher** будь-яким рядковим значенням і виведіть її значення в консоль. Результат має бути у верхньому регістрі. 7 Створіть приватну (“hard private”) числову властивість **id**. • Внесіть зміни до конструктора для ініціалізації цієї властивості. • Додайте метод **getID**(), який повинен повертати значення властивості **id**. • Виведіть об'єкт у консоль. • Викличте метод **getID**(). 8 Створіть статичну рядкову властивість **department** і проініціалізуйте її будь-яким значенням за замовчуванням. Внесіть зміни до методу **printItem**() – метод повинен додатково	properties through the constructor parameters **title - public, year - private**. 6 Create a private (“soft private”) string property **_publisher**. • Add a **publisher** getter that should return the value of the **_publisher** property in uppercase. • Add a **publisher** setter that should accept a string parameter **newPublisher** and sets the value of the **_publisher** property to the value of this parameter. • Initialize the **ref.publisher** property to any string value and print its value to the console. The result must be in uppercase. 7 Create a hard private numeric **id** property. • Modify the constructor to initialize this property. • Add a **getID**() method that should return the value of the **id** property. • Print the object to the console. • Call the **getID**() method. 8 Create a static string property **department** and initialize it to any default value. Make changes to the **printItem**() method - the method should	параметры конструктора **title – public, year – private**. 6 Создайте приватное (soft private) строковое свойство **_publisher**. • Добавьте гетер **publisher**, который должен возвращать значение **_publisher** свойства в верхнем регистре. • Добавьте сеттер **publisher**, который должен принимать строковый параметр **newPublisher** и устанавливает значение свойства **publisher** в значение этого параметра. • Проинициализируйте свойство **ref.publisher** любым строчным значением и выведите его значение в консоль. Результат должен быть в верхнем регистре. 7 Создайте приватное (“hard private”) числовое свойство **id**. • Внесите изменения в конструктор для инициализации этого свойства. • Добавте метод **getID()**, который должен возвращать значение свойства **id**. • Выведите объект в консоль. • Вызовите метод **getID()**. 8 Создайте статическое строчное свойство **department** и проинициализируйте его любым значением по умолчанию. Внесите изменения в метод **printItem()** – метод должен

виводити значення цієї статичної властивості у консоль.	additionally print the value of this static property to the console.	дополнительно выводить значение этого статического свойства в консоль.

Завдання 05.02. Розширення класів	**Task 05.02. Extending Classes**	**Задание 05.02. Расширение классов**
1 Створіть клас **Encyclopedia** як похідний клас від **ReferenceItem**. Додайте одну додаткову числову публічну властивість **edition**. Використайте параметри конструктора. 2 Об’явіть змінну **refBook** та створіть об'єкт **Encyclopedia**. Викличте метод **printItem**(); 3 Перевизначте метод **printItem**(). Додайте ключове слово **override**. Метод повинен виконувати те, що виконував раніше та додатково повинен виводити рядок у консоль «**Edition: edition (year)**». Ви отримаєте помилку, що властивість **year** недоступна. Щоб властивість стала доступна, змініть модифікатор доступу в класі **ReferenceItem** з **private** на **protected**.	1 Create the **Encyclopedia** class as a derived class from **ReferenceItem**. Add one additional numeric public property **edition**. Use constructor parameters. 2 Declare the **refBook** variable and create an **Encyclopedia** object. Call the **printItem()** method; 3 Override the **printItem**() method. Add the **override** keyword. The method should do what it did and additionally should output the string **"Edition: edition (year)"** to the console. You will get an error that the **year** property is not available. To make it available, change the access modifier in the **ReferenceItem** class to **protected**.	1 Создайте класс **Encyclopedia** как производный класс от **ReferenceItem**. 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().** Добавьте ключевое слово **override**. Метод должен делать то, что он делал и дополнительно должен выводит строчку в консоль «**Edition: edition (year)**». Вы получите ошибку, что свойство **year** недоступно. Чтобы оно было доступно измените модификатор доступа в классе **ReferenceItem** на **protected.**

Завдання 05.03. Абстрактні класи	**Task 05.03. Abstract Classes**	**Задание 05.03. Абстрактные классы**
1 Внесіть зміни до класу **ReferenceItem** – зробіть його абстрактним.	1 Change the **ReferenceItem** class to make it abstract.	1 Внесите изменения в класс **ReferenceItem** – сделайте его абстрактным.
2 Створіть абстрактний метод **printCitation**(), який повинен не приймати параметрів і не повертати значення. Цей метод має бути без реалізації. Після цього Ви отримаєте помилку в класі **Encyclopedia**, яка повідомлятиме, що не реалізовано абстрактний метод.	2 Create an abstract **printCitation**() method that should take no parameters and return no value. This method should not have an implementation. After that, you will get an error in the **Encyclopedia** class saying that the abstract method is not implemented.	2 Создайте абстрактный метод **printCitation()**, который должен не принимать параметров и не возвращать 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 Створіть метод **printCitation()** у класі **Encyclopedia**. Метод повинен виводити в консоль рядок "**title - year**".	3 Create a **printCitation()** method in the **Encyclopedia** class. The method should output the line **"title - year"** to the console.	3 Создайте метод **printCitation()** в классе **Encyclopedia**. Метод должен выводить в консоль строчку **«title – year»**.
4 Об’явіть змінну **refBook** та проініціалізуйте її об'єктом **Encyclopedia**. Викличте метод **printCitation**().	4 Declare a **refBook** variable and create an **Encyclopedia** object. Call the **printCitation()** method.	4 Объявите переменную **refBook** и создайте объект **Encyclopedia**. Вызовите метод **printCitation().**

Завдання 05.04. Реалізація інтерфейсів класами	**Task 05.04. Implementing Interfaces by Classes**	**Задание 05.04. Реализация интерфейсов классами**
1 Створіть клас **UniversityLibrarian**, який повинен реалізовувати інтерфейс **Librarian** та реалізуйте всі необхідні властивості. Метод **assistCustomer()** повинен виводити в консоль рядок **`${this.name} is assisting ${custName} with book ${bookTitle}`**. 2 Об’явіть змінну **favoriteLibrarian** за допомогою інтерфейсу **Librarian** і проініціалізуйте її за допомогою об'єкта, створеного класом **UniversityLibrarian**. Жодних помилок при цьому не повинно виникати. Проініціалізуйте властивість **name** та викличте метод **assistCustomer**().	1 Create a **UniversityLibrarian** class that should implement the **Librarian** interface and implement all required properties. The **assistCustomer**() method should output the string **`${this.name} is assisting ${custName} with the book ${bookTitle}`** to the console. 2 Declare a **favoriteLibrarian** variable using the **Librarian** interface and initialize it with an object created by the **UniversityLibrarian** class. No errors should be generated. Initialize the **name** property and call the **assistCustomer**() method.	1 Создайте класс **UniversityLibrarian**, который должен реализовывать интерфейс **Librarian** и реализуйте все необходимые свойства. Метод **assistCustomer()** должен выводить в консоль строчку **`${this.name} is assisting ${custName} with the book ${bookTitle}`.** 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()**.

Завдання 05.05. Перетин та об’єднання типів	**Task 05.05. Intersection and Union Types**	**Задание 05.05. Пересечение и объединение типов**
1 Створіть тип **PersonBook**. Використовуйте для цього інтерфейси **Person**, **Book** та перетин типів.	1 Create a **PersonBook** type. Use the **Person**, **Book** interfaces and type intersection for this.	1 Создайте тип **PersonBook**. Используйте для этого интерфейсы **Person**, **Book** и пересечение типов**.**
2 Об’явіть змінну з типом **PersonBook**, проініціалізуйте її літералом, виведіть її в консоль.	2 Declare a variable of type **PersonBook**, initialize it with a literal, print it to the console.	2 Объявите переменную с типом **PersonBook**, проинициализируйте ее литералом, выведите ее в консоль.
3 Створіть тип **BookOrUndefined**. Використовуйте для цього об'єднання інтерфейсу **Book** та **undefined**.	3 Create a **BookOrUndefined** type. Use the union of the **Book** interface and **undefined** for this.	3 Создайте тип **BookOrUndefined**. Используйте для этого объединение интерфейса **Book** и **undefined**.
4 Замініть тип значення, що повертається у функції **getBookByID()** на **BookOrUndefined**.	4 Change the return type in the **getBookByID**() function to **BookOrUndefined**.	4 Замените тип возвращаемого значения в функции **getBookByID()** на **BookOrUndefined**.
5 Створіть функцію **setDefaultConfig**(), яка приймає об'єкт **options**. Тип для об'єкта **TOptions** об’явіть за допомогою інтерфейса з необов'язковими числовими властивостями **duration** і **speed**. Функція повинна встановлювати значення властивостей за замовчуванням якщо вони не мають ніякого значення, використовуючи логічний оператор налового присвоєння та повертати об'єкт.	5 Create a **setDefaultConfig**() function that takes an **options** object. Declare the type for a **TOptions** object with an interface with optional numeric properties **duration** and **speed**. The function must set the default property values if they do not contain any value using the logical nullish coalescing operator and return an object.	5 Создайте функцию **setDefaultConfig()**, которая принимает объект **options**. Тип для объекта **TOptions** объявите с помощью интерфейса с необязательными числовыми свойствами **duration** и **speed**. Функция должна устанавливать значения свойств по-умолчанию, если они не содержат никакого значения, используя логический оператор налового присваивания, и возвращать объект.

06. Modules and Namespaces

Завдання 06.01. Використання простору імен	**Task 06.01. Namespaces Usage**	**Задание 06.01. Использование пространства имен**
1 Створіть папку для нового проекту **NamespaceDemo.**	1 Create a folder for the new project **NamespaceDemo.**	1 Создайте папку для нового проекта **NamespaceDemo.**
2 Створіть файл **utility-functions.ts.**	2 Create **utility-functions.ts** file.	2 Создайте файл **utility-functions.ts.**
3 Створіть простір імен **Utility.**	3 Create the **Utility** namespace.	3 Создайте пространство имен **Utility.**
4 Створіть та експортуйте вкладений простір імен **Fees.**	4 Create and export nested **Fees** namespace/	4 Создайте и экспортируйте вложенное пространство имен **Fees.**
5 Створіть та експортуйте функцію **calculateLateFee**() з вкладеного простору імен, яка приймає числовий параметр **daysLate** та повертає **fee**, обчислене як **daysLate * 0.25.**	5 Create and export a function **calculateLateFee**() from the nested namespace that takes a numeric parameter **daysLate** and returns a fee calculated as **daysLate * 0.25.**	5 Создайте и экспортируйте функцию **calculateLateFee()** из вложенного пространства имен, которая принимает числовой параметр **daysLate** и возвращает **fee**, вычисленное как **daysLate * 0.25.**
6 Створіть та експортуйте функцію **maxBooksAllowed**() з простору імен **Utility**, яка приймає один числовий параметр **age**. Якщо **age < 12**, то повертає **3,** інакше **10**.	6 Create and export a **maxBooksAllowed**() function from the **Utility** namespace that takes a single numeric parameter **age**. If **age < 12** then returns **3** otherwise **10**.	6 Создайте и экспортируйте функцию **maxBooksAllowed()** из пространства имен **Utility**, которая принимает один числовой параметр **age**. Если **age < 12**, то возвращает **3** иначе **10**.

7
Створіть функцію **privateFunc**(), яка виводить у консоль повідомлення **«This is a private function»**
 	
8
Створіть файл **app.ts**. В ньому напишіть фрагмент коду, який використовує функції із простору імен. Використайте ключове слово **import** та оголосіть аліас **util** для вкладеного простору імен **import util = Utility.Fees.**
 	
9
Запустіть компілятор та скомпілюйте лише **tsc app.ts --target ES5**. Ви отримаєте помилку. Додайте посилання на файл **utility-functions.ts.**
 	
10
Створіть **index.html** Скористайтеся наступним фрагментом HTML:
`<html>`
`<head></head>`
`<body>`
`<script src="utility-functions.js"></script>`
`<script src="app.js"></script>`
`</body>`
`</html>`
Запустіть компілятор ще раз і вкажіть опцію **--outFile bundle.js.** Підключіть отриманий файл до **index.html.**
	7
	Create a **privateFunc**() function that prints the message **"This is a private function"** to the console
	
	8
	Create an **app.ts** file. Write a code snippet in it that uses functions from the namespace. Use the **import** keyword and declare an alias **util** for the nested namespace. **import util = Utility Fees.**
	
	9
	Run the compiler and compile only **tsc app.ts --target ES5**. You will get an error. Add a link to the **utility-functions.ts** file.
	
	10
	Create **index.html** Use the following HTML snippet:
	<html>
	<head></head>
	<body>
	<script src="utility-functions.js"></script>
	<script src="app.js"></script>
	</body>
	</html>
	Run the compiler again and specify the **--outFile bundle.js** option. Include the resulting file in **index.html.**
		7
		Создайте функцию **privateFunc**(), которая выводит в консоль сообщение **«This is a private function»**
		
		8
		Создайте файл **app.ts.** Напишите в нем фрагмент кода, который использует функции из пространства имен. Используйте ключевое слово **import** и объявите алиас **util** для вложенного пространства имен. **import util = Utility.Fees.**
		
		9
		Запустите компилятор и скомпилируйте только **tsc app.ts --target ES5**. Вы получите ошибку. Добавьте ссылку на файл **utility-functions.ts.**
		
		10
		Создайте **index.html** Воспользуйтесь следующим фрагментом HTML:
		`<html>`
		`<head></head>`
		`<body>`
		`<script src="utility-functions.js"></script>`
		`<script src="app.js"></script>`
		`</body>`
		`</html>`
		Запустите еще раз компилятор и укажите опцию **--outFile bundle.js.** Подключите полученный файл в **index.html.**

Завдання 06.02. Експорт та імпорт	**Task 06.02. Export and Impot**	**Задание 06.02. Экспорт и импорт**
1 Створіть файл **enums.ts**, перенесіть до нього **enum Category**. Додайте експорт в кінці файлу.	1 Create an **enums.ts** file, move the **enum Category** to it. Add an export at the end of the file.	1 Создайте файл **enums.ts**, перенесите в него **enum Category**. Добавьте экспорт в конце файла.
2 Створіть файл **interfaces.ts** та • перенесіть до нього інтерфейси: **Book, DamageLogger, Person, Author, Librarian** • додайте імпорт **Category** • додайте експорт інтерфейсів **Book, DamageLogger, Person, Author, Librarian, TOptions** в кінці файлу. експортуйте **DamageLogger** під назвою **Logger**	2 Create an **interfaces.ts** file and • move the **Book**, **DamageLogger, Person, Author, Librarian** interfaces to it • add **Category** import • add the export of the interfaces **Book, DamageLogger, Person, Author, Librarian, TOptions** at the end of the file. • export the **DamageLogger** with new name - **Logger**	2 Создайте файл **interfaces.ts** и • перенесите в него интерфейсы **Book, DamageLogger, Person, Author, Librarian** • добавьте импорт **Category** • добавьте экспорт интерфейсов **Book, DamageLogger, Person, Author, Librarian, TOptions** в конце файла. экспортируйте **DamageLogger** с именем **Logger**
3 Створіть файл **classes.ts** та перенесіть до нього класи: **UniversityLibrarian, ReferenceItem**. • Додайте імпорт інтерфейсів як цілого модуля з ім'ям **Interfaces** • Змініть опис класу **UniversityLibrarian**, щоб він реалізовував інтерфейс **Interfaces.Librarian** • Додайте експорт в кінці файлу та експортуйте обидва класи.	3 Create a **classes.ts** file and move the following classes to it: **UniversityLibrarian, ReferenceItem**. • Add imports of interfaces as a whole module with name **Interfaces** • Change the definition of the **UniversityLibrarian** class to implement the **Interfaces.Librarian** interface. • Add an export at the end of the file and export both classes.	3 Создайте файл **classes.ts** и перенесите в него классы: **UniversityLibrarian, ReferenceItem**. • Добавьте импорт интерфейсов как целого модуля с именем **Interfaces** • Измените описание класса **UniversityLibrarian**, чтобы он реализовывал интерфейс **Interfaces.Librarian** • Добавьте экспорт в конце файла и экспортируйте оба класса.
4 Створіть файл **types.ts** і перенесіть у нього типи: **BookProperties, PersonBook, BookOrUndefined**.	4 Create a **types.ts** file and move the following types to it: **BookProperties, PersonBook, BookOrUndefined**.	4 Создайте файл **types.ts** и перенесите в него типы: **BookProperties, PersonBook, BookOrUndefined**.

• Додайте імпорт інтерфейсів **Book** та **Person** • Експортуйте типи із модуля. 5 Створіть файл **functions.ts** та перенесіть усі функції до нього. • Додайте імпорт інтерфейсу **Book**, **enum Category**, типів **BookProperties, BookOrUndefined** • Додайте експорт всіх функцій (не обов'язково) 6 Внесіть зміни до файлу **app.ts** • Додайте імпорт категорій, інтерфейсів **Book, Logger, Author, Librarian**, класів **UniversityLibrarian, ReferenceItem**, типу **PersonBook** та всіх функцій. • Змініть тип змінної **logDamage** на **Logger** (Завдання 04.02)	• Add import of the **Book** and **Person** interfaces • Export types from a module. 5 Create a **functions.ts** file and move all functions to it. • Add import of the **Book** interface, **Category** enum, **BookProperties, BookOrUndefined** types • Add export of all functions (optional) 6 Make changes to the **app.ts** file • Add import for **Category, Book, Logger, Author, Librarian** interfaces, **UniversityLibrarian, ReferenceItem** classes, **PersonBook** type, and all functions. • Change the type of the variable **logDamage** to **Logger** (Task 04.02)	• Добавьте импорт интерфейсов **Book** и **Person** • Экспортируйте типы из модуля. 5 Создайте файл **functions.ts** и перенесите все функции в него. • Добавьте импорт интерфейса **Book**, перечисления **Category**, типов **BookProperties**, **BookOrUndefined** • Добавьте экспорт всех функций (не обязательно) 6 Внесите изменения в файл **app.ts** • Добавьте импорт **Category**, интерфейсов **Book, Logger, Author, Librarian**, классов **UniversityLibrarian, ReferenceItem,** тип **PersonBook** и всех функций. • Измените тип переменной **logDamage** на **Logger (Задание 04.02)**

Завдання 06.03. Експорт за замовчуванням	**Task 06.03. Default Export**	**Задание 06.03. Экспорт по умолчанию**
1 Створіть файл **encyclopedia.ts** та перемістіть до нього клас **Encyclopedia**. Додайте імпорт **ReferenceItem**. 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**	1 Create an **encyclopedia.ts** file and move the **Encyclopedia** class into it. Add the **ReferenceItem** import. Add a default export. 2 Import this class into the **app.ts** as **RefBook.** 3 Make changes to the task **Task 05.02.** 4 Author: Yevhen_Zakharevych@epam.com Create a condition assertion function **assertRefBookInstance** in the **functions.ts** module. The function should accept **condition: any** and return the **asserts condition** type. If the condition is not met, then the function should throw an exception **"It is not an instance of RefBook"**. 5 Create and export a **printRefBook(data: any): void** function that uses the **assertRefBookInstance** function and calls the **printItem()** method on the **RefBook** instance. Set the test condition using the **instanceof** operator	1 Создайте файл **encyclopedia.ts** и переместите в него класс **Encyclopedia**. Добавьте импорт **ReferenceItem**. Добавьте экспорт по умолчанию. 2 Импортируйте данный класс в приложение как **RefBook.** 3 Внесите изменения в код задания **Task 05.02**. 4 Автор: Yevhen_Zakharevych@epam.com Создайте функцию-утверждения условия **assertRefBookInstance** в модуле **functions.ts.** Функция должна принимать **condition: any**, возвращать тип **asserts condition.** Если условие не выполняется, то функция должна бросать исключение «It is not an instance of RefBook»**.** 5 Создайте и экспортируйте функцию **printRefBook(data: any): void**, которая использует функцию **assertRefBookInstance** и вызывает метод **printItem()** у экземпляра **RefBook.** Условие проверки задать с помощью 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.**	6 Import the **printRefBook** function into your **app.ts** and call it on an instance of the **RefBook** class. 7 Create an instance of the **UniversityLibrarian** class and call the **printRefBook** function on it again.	6 Импортируйте функцию **printRefBook** в приложение и вызовите для экземпляра класса **RefBook**. 7 Создайте экземпляр класса **UniversityLibrarian** и снова вызовите для него функцию **printRefBook.**

Завдання 06.04. Реекспорт	**Task 06.04. Re-Export**	**Задание 06.04. Реэкспорт**
1 Створіть папку **classes** і перемістіть файл **encyclopedia.ts** до неї.	1 Create a **classes** folder and move the **encyclopedia.ts** file into it.	1 Создайте папку **classes** и переместите в нее файл **encyclopedia.ts.**
2 Рознесіть класи **UniversityLibrarian** і **ReferenceItem** по різних файлах і перемістіть в папку **classes**.	2 Separate the **UniversityLibrarian** and **ReferenceItem** classes into different files and move them to the **classes** folder.	2 Разнесите классы **UniversityLibrarian** и **ReferenceItem** по разным файлам и тоже переместите в папку **classes**.
3 Видаліть файл **classes.ts.**	3 Delete the **classes.ts** file.	3 Удалите файл **classes.ts.**
4 Створіть файл **classes/index.ts** і додайте до нього реекспорт класів **ReferenceItem, Encyclopedia** , використовуючи конструкцію **export ***, **export { default as …}**, а також додайте реекспорт класу **UniversityLibrarian**, використовуючи конструкцію **export * as UL**.	4 Create a file **classes/index.ts** and re-export the **ReferenceItem, Encyclopedia** classes using the **export *, export { default as ... }** construct, and re-export the **UniversityLibrarian** class using the **export * as UL** construct.	4 Создайте файл **classes/index.ts** и добавьте в него реэкспорт классов **ReferenceItem, Encyclopedia**, используя конструкцию **export ***, **export { default as … }**, а также добавьте реэкспорт класса **UniversityLibrarian**, используя конструкцию **export * as UL**.
5 Виправте імпорти у файлі **app.ts.**	5 Correct the imports in the **app.ts** file.	5 Исправьте импорты в файле **app.ts.**
6 Виправте створення екземпляра класу **UniversityLibrarian** у завданні **05.04**. та **06.03.**	6 Correct the creation of an instance of the **UniversityLibrarian** class in task **05.04.** and **06.03.**	6 Исправьте создание экземпляра класса **UniversityLibrarian** в задании **05.04**. и **06.03.**

Завдання 06.05. Вираз динамічного імпорту	**Task 06.05. Dynamic import expression**	**Задание 06.05. Выражение динамического импорта**

1
Створіть у папці **classes** файл **reader.ts** та реалізуйте клас **Reader**, який містить такі властивості:
• name: string;
• books: Book[] = [];
• 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 приймає значення **true**.
 	
4
Додайте до **webpack.config.js** об'єкт
`experiments: {`
`  topLevelAwait: true`
`}`
 	
5
Створіть екземпляр класу **Reader**. Виведіть його в консоль.
	1
	Create a **reader.ts** file in the **classes** folder and implement a **Reader** class that contains the following properties:
	• name: string;
	• books: Book[] = [];
	• take(book: Book): void - the method adds a book to the array of books.
	
	2
	Make changes to the **classes/index.ts** file, add a new module.
	
	3
	Implement a dynamic import expression using a **top level await/Promise** expression to load everything from the **'./classes'** path as a module. Implement loading under the condition that some variable gets the value **true**.
	
	4
	Add an object to **webpack.config.js**
	experiments: {
	topLevelAwait: true
	}
	
	5
	Create an instance of the **Reader** class. Output it to the console.
		1
		Создайте в папке **classes** файл **reader.ts** и реализуйте класс **Reader**, который содержит следующие свойства:
		• name: string;
		• books: Book[] = [];
		• 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 с использованием выражения **top level await/Promise** для загрузки всего из пути **'./classes'** как модуля. Загрузку реализовать при условии, если некоторый переменная получает значение **true**.
		
		4
		Добавьте в **webpack.config.js** объект
		experiments: {
		`  topLevelAwait: true`
		}
		
		5
		Создайте экземпляр класса **Reader**. Выведите его в консоль.

Завдання 06.06. Імпорт та експорт типів	**Task 06.06. Type-only imports and exports**	**Задание 06.06. Импорт и экспорт типов**
1 Створіть у папці **classes** файл **library.ts** та реалізуйте клас **Library**, який містить наступні властивості: • Id: number • name: string • address: string	1 Create a **library.ts** file in the **classes** folder and implement the **Library** class, which contains the following properties: • ID: number • name:string • address:string	1 Создайте в папке **classes** файл **library.ts** и реализуйте класс **Library**, который содержит следующие свойства: • Id: number • name: string • address: string
2 Внесіть зміни до файлу **classes/index.ts**. Експортуйте тип **Library**. Використовуйте конструкцію **export type {…}.**	2 Make changes to the **classes/index.ts** file. Export the **Library** type. Use the **export type {…}** construct.	2 Внесите изменения в файл **classes/index.ts**. Экспортируйте тип **Library**. Используйте конструкцию **export type {…}.**
3 Імпортуйте **Library** в **app.ts**. Оголосіть змінну за допомогою **Library**.	3 Import the **Library** type in **app.ts**. Declare a variable using the **Library** type.	3 Импортируйте тип **Library** в **app.ts**. Объявите переменную, используя тип **Library**.
4 Створіть екземпляр класу **Library**. Ви повинні отримати помилку. Закоментуйте рядок.	4 Create an instance of the **Library** class. You should get an error. Comment out the line.	4 Создайте экземпляр класса **Library**. Вы должны получить ошибку. Закомментируйте строчку.
5 Об’явіть змінну, вкажіть тип **Library**. Проініціалізуйте літералом, виведіть у консоль.	5 Declare a variable, annotate it with the **Library** type. Initialize with a literal, output to the console.	5 Объявите переменную, укажите тип **Library**. Проинициализируйте литералом, выведите в консоль.

07. Generics

Завдання 07.01. Загальні функції	**Task 07.01. Generic functions**	**Задание 07.01. Общие функции**
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. Експортуйте цю функцію.	1 In the **functions.ts** file, create a generic function **purge()** that takes one parameter, a generic **inventory** array, and returns a generic array of the same type that contains the elements of the original array minus the first two elements. Export this function.	1 Создайте в файле **functions.ts** дженерик (общую) 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 Імпортуйте функцію **purge()** у **app.ts**.	2 Import the **purge()** function into **app.ts**.	2 Импортируйте функцию **purge()** в **app.ts**.
3 Додайте категорію **Software** у файл **enums.ts**.	3 Add the **Software** category in the **enums.ts** file.	3 Добавьте категорию **Software** в файле **enums.ts**.
4 Об’явіть 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 },	4 Declare an **inventory** variable that contains the following array of books [{ 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 },	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 Викличте функцію **purge**() та передайте їй ці дані. Виведіть результат у консоль. 6 Викличте функцію **purge()** з числовим масивом і знову виведіть результат у консоль. 7 Об’явіть змінну **purgeNumbers** та присвойте їй функцію **purge** зі значенням параметру типу **number**. Викличте функцію **purgeNumbers**() та передайте їй числовий масив та масив рядків. 8 Додайте **in/out/in out** до параметру типу у функції **purge()**.	{ id: 13, title: 'Cool autoexec.bat Scripts!', author: 'C. D.', available: true, category: Category.Software }]; 5 Call the **purge()** function and pass this data to it. Print the result to the console. 6 Call the **purge()** function with a numeric array and print the result to the console again. 7 Declare a **purgeNumbers** variable and assign the **purge()** function to it with a value of type parameter **number**. Call the **purgeNumbers()** function and pass in an array of numbers and an array of strings. 8 Add **in/out/in out** to the type parameter in the **purge()** function.	{ 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 функцию **purge()** с числовым массивом и снова выведите результат в консоль. 7 Объявите переменную **purgeNumbers** и присвойте ей функцию **purge()** со значением параметра типа **number**. Вызовите функцию **purgeNumbers**() и передайте числовой массив и массив строк. 8 Добавьте **in/out/in out** к параметру типа в функции **purge().**

Завдання 07.02. Загальні інтерфейси і класи	**Task 07.02. Generic interfaces and classes**	**Задание 07.02. Общие интерфейсы и классы**
1 Створіть інтерфейс **Magazine**, який містить дві рядкові властивості, **title**, **publisher** та додайте його у файл **interfaces.ts**. Експортуйте цей інтерфейс.	1 Create a **Magazine** interface that contains two string properties **title**, **publisher** and add it to the **interfaces.ts** file. Export this interface.	1 Создайте интерфейс **Magazine**, который содержит два строчных свойства **title**, **publisher** и добавьте его в файл **interfaces.ts**. Экспортируйте данный интерфейс.
2 Створіть файл **classes/shelf.ts** і, використовуючи експорт за замовчуванням, реалізуйте дженерик клас **Shelf**: • додайте приватну властивість **items**, яка є масивом елементів типу T. • додайте метод **add**(), який приймає один параметр **item** типу T і додає його в масив. Нічого не повертає. • додайте метод **getFirst**(), який нічого не приймає, і повертає перший елемент із **items**.	2 Create a file **classes/shelf.ts** and use the default export to implement the generic **Shelf** class: • add the private property **items**, which is an array of elements of type T. • add an **add()** method that takes a single item parameter of type T and adds it to the array. Returns nothing. • add a **getFirst()** method that takes nothing but returns the first item from the shelf.	2 Создайте файл **classes/shelf.ts** и используя экспорт по умолчанию реализуйте дженерик класс **Shelf**: • добавьте приватное свойство **items**, которое является массивом элементов типа T. • добавьте метод **add()**, который принимает один параметр **item** типа T и добавляет его в массив. Ничего не возвращает. • добавьте метод **getFirst()**, который ничего не принимает, а возвращает первый элемент с полки.
3 Додайте реекспорт у файл **classes/index.ts**	3 Add re-export to **classes/index.ts** file	3 Добавьте реэкспорт в файл **classes/index.ts**
4 Імпортуйте клас **Shelf** і інтерфейс **Magazine** в **app.ts**.	4 Import the **Shelf** class and the **Magazine** interface in **app.ts**.	4 Импортируйте класс **Shelf** и интерфейс **Magazine** в **app.ts**.
5 Закоментуйте код, який відноситься до функції **purge**(), крім змінної **inventory.**	5 Comment out the code related to the **purge()** function, except the **inventory** variable.	5 Закомментируйте код, который относится к функции **purge()**, кроме переменной **inventory.**
6	6	6

Створіть екземпляр класу **Shelf** - **bookShelf** і збережіть усі книжки з **inventory** в **bookShelf**. Отримайте першу книжку і виведіть її назву в консоль.
 	
7
Об'явіть змінну **magazines**, яка містить наступні дані:
`[`
`{ title: 'Programming Language Monthly', publisher: 'Code Mags' },`
`{ title: 'Literary Fiction Quarterly', publisher: 'College Press' },`
`{ title: 'Five Points', publisher: 'GSU' }`
`];`
 	
8
Створіть екземпляр класу **Shelf** - **magazineShelf** і збережіть усі журнали в **magazineShelf**. Отримайте перший журнал і виведіть його в консоль.
	Create an instance of the **Shelf** class - **bookShelf** and store all books from **inventory** in **bookShelf**. Get the first book and print its title to the console.
	
	7
	Declare a variable **magazines** that contains the following data:
	`[`
	`{ title: 'Programming Language Monthly', publisher: 'Code Mags' },`
	`{ title: 'Literary Fiction Quarterly', publisher: 'College Press' },`
	`{ title: 'Five Points', publisher: 'GSU' }`
	`];`
	
	8
	Create an instance of the **Shelf** - **magazineShelf** class and store all magazines in **magazineShelf**. Get the first log and print it to the console.
		Создайте экземпляр класса **Shelf** - **bookShelf** и сохраните все книжки из **inventory** в **bookShelf**. 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 экземпляр класса **Shelf** - **magazineShelf** и сохраните все журналы в **magazineShelf**. Получите первый журнал и выведите его в консоль.

Завдання 07.03. Загальні обмеження

1
Внесіть зміни в клас **Shelf**:

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

Після додавання цих методів ви отримаєте помилку - властивість **title** не існує на типі T.

2
У файлі **interfaces.ts** створіть інтерфейс **ShelfItem**, який повинен містити всі необхідні властивості, які повинен мати тип T, а саме **title**.

3
Додайте загальне обмеження для класу, розширив тип T від нього.

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 для першого параметру, щоб значення були об'єктами. Додайте обмеження для другого параметру, щоб значення були

Task 07.03. Generic constraints

1
Make changes to the **Shelf** class:

- add a **find()** method that takes a string parameter **title** and returns the first element found on the shelf of type T.
- add a **printTitles()** method that prints the titles of items on the shelf to the console.

After adding these methods, you will get an error - **title** property does not exist on type T.

2
In the **interfaces.ts** file, create the interface **ShelfItem**, which should contain all the necessary properties that type T should have, namely **title**.

3
Add a generic constraint to the class and extend the type T from it.

4
Call the **printTitles()** function for the magazines.

5
Find the **'Five Points'** magazine and print it to the console.

6
Create a **getObjectProperty**() function. Add two parameters of type **TObject**, **TKey**. Add a constraint for the first parameter so that the values will be objects. Add a constraint for the second parameter so that the values will be only

Задание 07.03. Общие ограничения

1
Внесите изменения в класс **Shelf**:

- добавьте метод **find()**, который принимает строчный параметр **title** и возвращает первый найденный элемент на полке типа T.
- добавьте метод **printTitles()**, который выводит в консоль наименования того, что находится на полке.

После добавления этих методов вы получите ошибку - свойство **title** не существует на типе T.

2
В файле **interfaces.ts** создайте интерфейс **ShelfItem**, который должен содержать все необходимые свойства, которые должен иметь тип T, а именно **title**.

3
Добавьте общее ограничение для класса расширив тип T от него.

4
Вызовите функцию **printTitles()** для журналов.

5
Найдите журнал **'Five Points'** и выведите его в консоль.

6
Создайте функцию **getObjectProperty()**. Добавьте два параметра типа **TObject**, **TKey**. Добавьте ограничение для первого параметра, чтобы значения были объектами. Добавьте ограничение для второго параметра, чтобы

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.	keys of an object of type **TObject** using the **keyof** operator. Change the return type to **TObject[TKey] | string.** The body of the function is similar to the body of the **getProperty**() function. Call this function.	значения были только ключами объекта типа **TObject**, используя **keyof** оператор. Измените тип возвращаемого значения на **TObject[TKey] | string**. Тело функции аналогично телу функции **getProperty()**. Вызовите эту функцию.

Завдання 07.04. Утиліти	**Task 07.04. Utilities**	**Задание 07.04. Утилиты**
1 Об'явіть аліас типу **BookRequiredFields** у файлі **types.ts**, використовуючи інтерфейс **Book** та утиліту **Required**.	1 Declare an alias for the **BookRequiredFields** type in **types.ts** using the **Book** interface and the **Required** utility.	1 Объявите алиас типа **BookRequiredFields** в файле **types.ts**, используя интерфейс **Book** и утилиту **Required**.
2 Об'явіть змінну **bookRequiredFields** типу **BookRequiredFields** та присвойте їй відповідний об'єкт.	2 Declare a variable **bookRequiredFields** of type **BookRequiredFields** and assign the appropriate object to it.	2 Объявите переменную **bookRequiredFields** типа **BookRequiredFields** и присвойте ей соответствующий объект.
3 Об'явіть аліас типу **UpdatedBook**, використовуючи інтерфейс **Book** та утиліту **Partial.**	3 Declare an alias of type **UpdatedBook** using the **Book** interface and the **Partial** utility.	3 Объявите алиас типа **UpdatedBook,** используя интерфейс **Book** и утилиту **Partial.**
4 Об'явіть змінну **updatedBook** типу **UpdatedBook** і присвойте їй відповідний об'єкт.	4 Declare an **updatedBook** variable of type **UpdatedBook** and assign the appropriate object to it.	4 Объявите переменную **updatedBook** типа **UpdatedBook** и присвойте ей соответствующий объект.
5 Об'явіть аліас типу **AuthorWoEmail**, використовуючи інтерфейс **Author** та утиліту **Omit**.	5 Declare an alias of type **AuthorWoEmail** using the **Author** interface and the **Omit** utility.	5 Объявите алиас типа **AuthorWoEmail,** используя интерфейс **Author** и утилиту **Omit**.
6 Об'явіть аліас **CreateCustomerFunctionType** для функціонального типу функції **createCustomer()**.	6 Declare an alias **CreateCustomerFunctionType** for the functional type of the **createCustomer**() function.	6 Объявите алиас **CreateCustomerFunctionType** для функционального типа функции **createCustomer().**
7 Об'явіть змінну **params**, використовуючи аліас типу **CreateCustomerFunctionType** і утиліту	7 Declare the **params** variable using the **CreateCustomerFunctionType** alias and the	7 Объявите переменную **params**, используя алиас типа **CreateCustomerFunctionType** и

Parameters, викличте функцію **createCustomer()**, передавши змінну **params**.	**Parameters** utility, call the **createCustomer**() function, passing the **params** variable.	утилиту **Parameters**, вызовите функцию **createCustomer(),** передав переменную **params**.

Завдання 07.05. Відображені типи, умовні типи	**Task 07.05. Mapped types, conditional types**	**Задание 07.05. Сопоставленные типы, условные типы**
1 Об'явіть у файлі **types.ts** аліас **fn** для функціонального типу функції, яка приймає три параметри з типами **string, number, boolean** і повертає тип **symbol**. 2 Об'явіть аліаси типів **Param1<T>, Param2<T>, Param3<T>**, які повертають тип першого, другого та третього параметрів функції відповідно. 3 Об'явіть 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 загальне обмеження для T розширивши його від типу **object** у **RequiredProps** та **OptionalProps**. 5 Об'явіть аліас типу **BookRequiredProps** та **BookOptionalProps**, використовуючи інтерфейс	1 Declare an alias **fn** in the file **types.ts** for the functional type of the function that takes three parameters with the types **string, number, boolean** and returns the type **symbol**. 2 Declare aliases of **Param1<T>, Param2<T>, Param3<T>** types that return the type of the first, second, and third function parameters, respectively. 3 Declare aliases **P1, P2, P3** and get the types of the first, second and third parameters of type **fn**. **Author**: Olena_Hlukhovska@epam.com 4 Create the **RequiredProps<T>** and **OptionalProps<T>** utilities in **types.ts** that return the union type of the object's **required** and **optional** properties. Use **mapped type** to iterate over the keys of T and **conditional type** to transform key values of type T. Add a generic constraint on T by extending it from type **object** to **RequiredProps** and **OptionalProps**. 5 Declare an alias of type **BookRequiredProps** and **BookOptionalProps** using the **Book** interface and	1 Объявите в файле **types.ts** алиас **fn** для функционального типа функции, которая принимает три параметра с типами **string, number, boolean** и возвращает тип **symbol**. 2 Объявите алиасы типов **Param1<T>, Param2<T>, Param3<T>** которые возвращают тип первого, второго и третьего параметра функции соответственно. 3 Объявите алиасы **P1**, **P2**, **P3** и получите типы первого, второго и третьего параметров типа **fn**. **Автор**: Olena_Hlukhovska@epam.com 4 Создайте утилиты **RequiredProps<T>** и **OptionalProps<T>** в файле **types.ts,** которые возвращают union тип **required** и **optional** свойств объекта. Используйте **mapped type** для перебора ключей **T** и **conditional type** для трансформации значений ключей типа **T**. Добавьте общее ограничение для **T** расширив его от типа **object** в **RequiredProps** и **OptionalProps**. 5 Объявите алиас типа **BookRequiredProps** и **BookOptionalProps**, используя interface **Book** и

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 Об'явіть аліас типу **BookRequiredPropsType** та **BookOptionalPropsType**, використовуючи інтерфейс **Book**, аліаси типу **BookRequiredProps** та **BookOptioalProps** та утиліту **RemoveProps** Спробуйте замість **Book** передати **Author**. Домашне завдання **Автор**: Oleksandr_Cherevach@epam.com 8 Створіть функцію **update()**, яка приймає один параметр типу **boolean**. Якщо значення аргументу **true**, функція повинна повертати значення типу **string**. Якщо значення аргументу **false**, функція повинна повертати значення типу **number**.	the **RequiredProps** and **OptionalProps** utilities. Try passing a primitive type instead of **Book**. 6 Create a **RemoveProps<T extends object, TProps extends keyof T>** utility that removes **TProps** properties from the passed type T. 7 Declare an alias of type **BookRequiredPropsType** and **BookOptionalPropsType** using the interface **Book**, aliases of type **BookRequiredProps** and **BookOptioalProps**, and the **RemoveProps** utility Try passing **Author** instead of **Book**. Homework **Author**: Oleksandr_Cherevach@epam.com 8 Create an **update**() function that takes one **boolean** parameter. If the argument value is **true**, then the function must return a value of type **string**. If the argument value is **false**, then the function must return a value of type **number**.	утилиты **RequiredProps** и **OptionalProps**. Попробуйте вместо **Book** передать примитивный type. 6 Создайте утилиту **RemoveProps<T extends object, TProps extends keyof T>,** которая удалаяет свойства **TProps** с переданого типа **T**. 7 Объявите алиас типа **BookRequiredPropsType** и **BookOptionalPropsType**, используя interface **Book**, алиасы типа **BookRequiredProps** и **BookOptioalProps** и утилиту **RemoveProps** Попробуйте вместо **Book** передать **Author**. Домашнее задание **Автор**: Oleksandr_Cherevach@epam.com 8 Создайте функцию **update()**, которая принимает один параметр типа **boolean**. Если значение аргумента **true**, то функция должна возвращать значение типа **string**. Если значение аргумента **false**, то функция должна возвращать значение типа **number**.

08. Decorators

Завдання 08.01. Декоратор класу	**Task 08.01. Class decorator**	**Задание 08.01. Декоратор класса**
1 Створіть файл **decorators.ts**.	1 Create a **decorators.ts** file.	1 Создайте файл **decorators.ts.**
2 Створіть декоратор класу **@freeze()**, 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 **"Freezing the constructor + параметр"**. Використовуйте метод **Object.freeze()**.	2 Create a **@freeze()** class decorator to prevent new properties from being added to the class object and object prototype. The decorator function must take one string parameter and should return nothing. Before executing the functionality, the function should print the message **“Freezing the constructor + parameter”** to the console. Use the **Object.freeze()** method.	2 Создайте декоратор класса **@freeze()**, 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 **«Freezing the constructor + параметр»**. Используйте метод **Object.freeze().**
2 Застосуйте цей декоратор до класу **UniversityLibrarian.**	3 Apply this decorator to the **UniversityLibrarian** class.	3 Примените данный декоратор к классу **UniversityLibrarian**.
3 Створіть екземпляр класу **UniversityLibrarian**. Перевірте повідомлення у консолі.	4 Create an instance of the **UniversityLibrarian** class. Check the message in the console.	4 Создайте экземпляр класса **UniversityLibrarian**. Проверьте сообщение в консоли.

Завдання 08.02. Декоратор класу	**Task 08.02. Class decorator**	**Задание 08.02. Декоратор класса**
1 Створіть декоратор класу **@logger()**, який змінюватиме конструктор класу.	1 Create an **@logger()** class decorator that will modify the class constructor.	1 Создайте декоратор класса **@logger()**, который будет изменять конструктор класса.
2 Об'явіть всередині декоратора змінну **newConstructor: Function** та проініціалізуйте її функціональним виразом. Новий конструктор повинен: • виводити в консоль повідомлення **"Creating new instance"** • виводити переданий параметр (ім'я класу). • створювати нову властивість **age** зі значенням **30**.	2 Declare a **newConstructor: Function** variable inside the decorator and initialize it with a function expression. The new constructor should: • print the message **"Creating new instance"** to the console • output the passed parameter (class name). • create a new property **age** with value **30**.	2 Объявите внутри декоратора переменную **newConstructor**: **Function** и проинициализируйте ее функциональным выражением. Новый конструктор должен: • выводить в консоль сообщение **«Creating new instance»** • выводить переданный параметр (имя класса). • создавать новое свойство **age** со значением **30**.
3 Проініціалізуйте прототип нового конструктора об'єктом, створеним на основі прототипу переданого класу, використовуючи **Object.create()** або **Object.setPrototypeOf()**.	3 Initialize the new constructor's prototype with an object created from the passed class's prototype using **Object.create()** or **Object.setPrototypeOf().**	3 Проинициализируйте прототип нового конструктора объектом, созданным на основе прототипа переданного класса используя **Object.create()** или **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}`**.	4 Add a new **printLibrarian()** method to the prototype of the new constructor, which should output the string **`Librarian name: ${this.name}, Librarian age: ${this.age}`** to the console.	4 Добавьте новый метод в прототип нового конструктора **printLibrarian()**, который должен выводить в консоль строку **`Librarian name: ${this.name}, Librarian age: ${this.age}`**.
5 Поверніть з декоратора новий конструктор, попередньо привівши його до типу **TFunction**.	5 Return a new constructor from the decorator, first nerrowing it to the **TFunction** type.	5 Верните из декоратора новый конструктор, предварительно приведя его к типу **TFunction**.
6	6	6

Застосуйте цей декоратор до класу **UniversityLibrarian**. Перевірте результат роботи в консолі. 7 Об'явіть змінну **fLibrarian** та створіть екземпляр класу **UniversityLibrarian**. Вкажіть значення **Anna** для **name**. Викличте метод **printLibrarian**().	Apply this decorator to the **UniversityLibrarian** class. Check the output in the console. 7 Declare a variable **fLibrarian** and create an instance of the **UniversityLibrarian** class. Set **name** to **Anna**. Call the **printLibrarian**() method.	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().**

Завдання 08.03. Декоратор методу	**Task 08.03. Method decorator**	**Задание 08.03. Декоратор метода**
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:** • **assistFaculty()** – виводить у консоль повідомлення **«Assisting faculty»**. • **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.	1 Create the **@writable()** method decorator as a factory that takes a boolean **isWritable** parameter. The decorator should set the descriptor's property writable to the passed value. 2 Add two methods to the **UniversityLibrarian** class: • **assistFaculty**() - prints the message **"Assisting faculty"** to the console. • **teachCommunity**() - prints the message **"Teaching community"** to the console. 3 Decorate the **assistFaculty**() method as **read/write** and the **teachCommunity**() method as **read**. 4 Try to change the methods of an instance of this class.	1 Создайте декоратор метода **@writable()** как фабрику, которая получает булевый параметр **isWritable**. Декоратор должен устанавливать свойство дескриптора **writable** в переданное значение. 2 Добавьте два метода для класса **UniversityLibrarian:** • **assistFaculty()** – выводит в консоль сообщение **«Assisting faculty»**. • **teachCommunity()** – выводит в консоль сообщение **«Teaching community».** 3 Задекорируйте метод **assistFaculty()** как изменяемый, а метод **teachCommunity()** как неизменяемый. 4 Попробуйте поменять методы у экземпляра этого класса.

Завдання 08.04. Декоратор методу	**Task 08.04. Method decorator**	**Задание 08.04. Декоратор метода**
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 браузера **window.confirm.**	1 Create the **@timeout()** method decorator as a factory that takes a numeric parameter - the number of milliseconds. The method to which the decorator is applied should run after the specified amount of time, and only if the user has given confirmation to it using the browser's **window.confirm** window.	1 Создайте декоратор метода **@timeout()** как фабрику, которая получает числовой параметр – количество миллисекунд. Метод, к которому применяется декоратор, должен запускаться через указанное количество времени и только, если пользователь дал на это согласие с помощью подтверждающего окна браузера **window.confirm.**
2 Декоратор повинен перевизначати властивість дескриптора **value**. Новий метод повинен використовувати **setTimout**() та запускати початковий метод через вказану кількість часу. Поверніть з декоратора новий дескриптор.	2 The decorator should override the **value** property of the descriptor. The new method should use **setTimout()** and run the original method after the specified amount of time. Return a new descriptor from the decorator.	2 Декоратор должен переопределять свойство дескриптора **value.** Новый метод должен использовать **setTimout()** и запускать первоначальный метод через указанное количество времени. Верните из декоратора новый дескриптор.
3 Застосуйте декоратор до методу **printItem()** класу **ReferenceItem**.	3 Apply a decorator to the **printItem()** method of the **ReferenceItem** class.	3 Примените декоратор к методу **printItem()** класса **ReferenceItem**.
4 Створіть екземпляр класу **Encyclopedia** та викличте метод **printItem**().	4 Create an instance of the **Encyclopedia** class and call the **printItem**() method.	4 Создайте экземпляр класса **Encyclopedia** и вызовите метод **printItem().**

Завдання 08.05. Декоратор параметра	**Task 08.05. Parameter decorator**	**Задание 08.05. Декоратор параметра**
1 Створіть декоратор параметра методу - **@logParameter()**, який повинен зберігати індекс параметра, до якого застосовується декоратор у властивість прототипу **${methodName}_decor_params_indexes**. Властивість організувати як масив.	1 Create a method parameter decorator - **@logParameter()**, which should save the index of the parameter to which the decorator is applied to the **${methodName}_decor_params_indexes** prototype property. Property should be organized as an array.	1 Создайте декоратор параметра метода - **@logParameter()**, который должен сохранять индекс параметра, к которому применяется декоратор в свойство прототипа **${methodName}_decor_params_indexes.** Свойство организовать в виде массива.
2 Створіть декоратор методу **@logMethod()**. Декоратор повинен перевизначати метод, до якого він застосовується та повертати новий дескриптор.	2 Create a **@logMethod()** method decorator. The decorator should override the method it is applied to and return a new descriptor.	2 Создайте декоратор метода **@logMethod().** Декоратор должен переопределять метод, к которому он применяется и возвращать новый дескриптор.
3 Перевизначений метод повинен отримати доступ до індексів, що знаходяться у властивості **${methodName}_decor_params_indexes** і для кожного параметра виводити його значення у форматі **Method: ${methodName}, ParamIndex: ${ParamIndex}, ParamValue: ${ParamValue}**	3 The overridden method should access the indexes in the **${methodName}_decor_params_indexes** property and output its value for each parameter in the format **Method: ${methodName}, ParamIndex: ${ParamIndex}, ParamValue: ${ParamValue}**	3 Переопределенный метод должен получить доступ к индексам, находящимся в свойстве **${methodName}_decor_params_indexes** и для каждого параметра выводить его значение в формате **Method: ${methodName}, ParamIndex: ${ParamIndex}, ParamValue: ${ParamValue}**
4 Задекоруйте метод **assistCustomer()** та всі його параметри відповідними декораторами.	4 Decorate the **assistCustomer()** method and all its parameters with the appropriate decorators.	4 Задекорируйте метод **assistCustomer()** и все его параметры соответствующими декораторами.
5 Створіть екземпляр класу **UniversityLibrarian**, проініціалізуйте властивість **name**, викличте метод **assistCustomer()**.	5 Create an instance of the **UniversityLibrarian** class, initialize the **name** property, call the **assistCustomer**() method.	5 Создайте экземпляр класса **UniversityLibrarian**, проинициализируйте свойство **name**, вызовите метод **assistCustomer()**.

Завдання 08.06. Декоратор властивості

1
Створіть фабричну функцію декоратора властивості **@format(pref:string = 'Mr./Mrs.')**, яка при застосуванні до властивості форматує її 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 {
            values.get(this);
          }
        },
        set(value: any) {
          if (setTransformer) {
```

Task 08.06. Property decorator

1
Create a property decorator factory function **@format(pref: string = 'Mr./Mrs.')** that, when applied to a property, formats its output by adding a **pref** prefix. The factory function should return a function with a property decorator signature, within which the **makeProperty(target, propertyName, value => `${pref} ${value}`, value => value)** function should be called.

2
The **makeProperty()** function looks like this:

```
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) {
```

Задание 08.06. Декоратор свойства

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) {
```

```
            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().** Створіть екземпляр класу **UniversityLibrarian**. Встановіть значення для властивості **name**, потім отримайте його та виведіть у консоль.

```
            values.set(this,
setTransformer(value));
          } else {
            values.set(this, value);
          }
        },
        enumerable: true
      });
      this[propertyName] = firstValue;
    },
    enumerable: true,
    configurable: true
  });
}
```

3

Add a **makeProperty()** function to **decorators.ts**

4

Decorate the **name** property of the **UniversityLibrarian** class with the **@format()** decorator. Create an instance of the **UniversityLibrarian** class. Set a value for the **name** property, then get it and print it to the console.

```
          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 **decorators.ts**

4

Задекорируйте свойство **name** класса **UniversityLibrarian** декоратором **@format().** Создайте экземпляр класса **UniversityLibrarian.** Установите значение для свойства **name**, затем получите его и выведите в консоль.

Завдання 08.07. Декоратор аксесорів	**Task 08.07. Accessors decorator**	**Задание 08.07. Декоратор аксесоров**
1 Створіть декоратор аксесору **@positiveInteger()**, який генерує виняток у випадку, якщо властивості встановлюється значення менше **1** і не ціле.	1 Create an **@positiveInteger()** accessor decorator that throws an exception if the property is set to a value less than **1** and not an integer.	1 Создайте декоратор аксессора **@positiveInteger()**, который генерирует исключение в случае, если свойству устанавливается значение менше **1** и не целое.
2 Додайте до класу **Encyclopedia** приватну числову властивість **_copies**, а також **геттер** і **сеттер** для цієї властивості, які повертають значення та встановлюють значення відповідно.	2 Add a private numeric property **_copies** to the **Encyclopedia** class, as well as a **getter** and **setter** for this property that return a value and set the value respectively.	2 Добавьте в класс **Encyclopedia** приватное числовое свойство **_copies,** а также **геттер** и **сеттер** для этого свойства, которые возвращают значение и устанавливают значение соответственно.
3 Задекоруйте гетер або сетер декоратором **@positiveInteger()**.	3 Decorate the getter or setter with the **@positiveInteger()** decorator.	3 Задекорируйте геттер или сеттер декоратором **@positiveInteger().**
4 Створіть екземпляр класу **Encyclopedia**. Спробуйте встановити різні значення **-10, 0, 4.5, 5**	4 Create an instance of the **Encyclopedia** class. Try to set different values, **-10, 0, 4.5, 5**	4 Создайте экземпляр класса **Encyclopedia**. Попробуйте установить разные значения, **-10, 0, 4.5, 5**

09. Asynchronous Patterns

Завдання 09.01. Функція зворотнього виклику	**Task 09.01. Callback function**	**Задание 09.01. Функция обратного вызова**
1 У файлі **interfaces.ts** створіть інтерфейс для функції зворотного виклику **LibMgrCallback**, яка приймає два параметри: • **err: Error | null,** • **titles: string[] | null** і нічого не повертає.	1 In the **interfaces.ts** file, create an interface **LibMgrCallback** for the callback function that takes two parameters: • **err: Error | null,** • **titles: string[] | null** and returns nothing.	1 В файле **interfaces.ts** создайте интерфейс для функции обратного вызова **LibMgrCallback**, которая принимает два параметра: • **err: Error | null,** • **titles: string[] | null** и ничего не возвращает.
2 У файлі **interfaces.ts** створіть дженерик інтерфейс для функції зворотнього виклику **Callback<T>**, яка приймає два параметри: • **err: Error | null,** • **data: T | null** і нічого не повертає.	2 In the **interfaces.ts** file, create a generic interface **Callback<T>** for the callback function that takes two parameters: • **err: Error | null,** • **data: T | null** and returns nothing.	2 В файле **interfaces.ts** создайте дженерик интерфейс для функции обратного вызова **Callback<T>,** которая принимает два параметра: • **err: Error | null,** • **data: T | null** и ничего не возвращает.
3 У файлі **functions.ts** створіть функцію **getBooksByCategory()**, яка приймає два параметри: • **category: Category** • **callback** – тип, раніше створений інтерфейс і нічого не повертає. Функція повинна використовувати **setTimeout**() та через 2с виконати наступний код: • у розділі **try**: використовувати функцію **getBookTitlesByCategory**() для отримання заголовків книг за категорією;	3 In your **functions.ts** file, create a **getBooksByCategory()** function that takes two parameters: • **category - Categories** • **callback** – type, previously created interface and returns nothing. The function should use **setTimeout()** and after 2s execute the following code: • in the **try** section: use the **getBookTitlesByCategory()** function to get book titles by category;	3 В файле **functions.ts** создайте функцию **getBooksByCategory()**, которая принимает два параметра: • **category:** Category • **callback** – тип, ранее созданный интерфейс и ничего не возвращает. Функция должна использовать **setTimeout()** и через 2с выполнить следующий код: • в секции **try**: использовать функцию **getBookTitlesByCategory()** для получения заголовков книг по категории;

• якщо знайшли книги, то викликати функцію зворотного виклику та передати: **null** та знайдені книги; • якщо не знайшли книг, то згенерувати виняток **throw new Error('No books found.');** • у секції **catch**: викликати функцію зворотного виклику та передати: **error** і **null**. 4 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. 5 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.	• if books are found, then call the callback function and pass: **null** and found books; • if no books found, then throw an exception **throw new Error('No books found.');** • in the **catch** section: call the callback function and pass: **error and null.** 4 Create a **logCategorySearch()** function that has the signature described in the **LibMgrCallback** or **Callback** interface. If an error object has arrived, then display the **err.message** property, otherwise display the titles of the books. 5 Call the **getBooksByCategory()** function and pass the necessary arguments to it. Add output to the console before and after calling this function. Use **Category.JavaScript** and **Category.Software** as the value of the first parameter.	• если нашли книги, то вызвать функцию обратного вызова и передать: **null** и найденные книги; • если не нашли книг, то сгенерировать исключение **throw new Error('No books found.')**; • в секции **catch**: вызвать функцию обратного вызова и передать: **error** и **null**. 4 Создайте функцию **logCategorySearch(),** которая имеет сигнатуру, описанную в интерфейсе **LibMgrCallback** или **Callback**. 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.

Завдання 09.02. Проміси	**Task 09.02. Promises**	**Задание 09.02. Промисы**
1 Створіть функцію **getBooksByCategoryPromise**(), яка приймає один параметр – **category** та повертає проміс – масив заголовків книг.	1 Create a **getBooksByCategoryPromise()** function that takes one parameter - **category** and returns a promise - an array of book titles.	1 Создайте функцию **getBooksByCategoryPromise()**, которая принимает один параметр – **category** и возвращает промис – массив заголовков 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.	2 Use **new Promise((resolve, reject) => { setTimeout(() => {…}, 2000) });** Add code similar to the **getBooksByCategory()** function, only now use **resolve()** and **reject()**. Return the created promise from the function.	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 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.	3 Call the **getBooksByCategoryPromise()** function and register callback functions with the **then** and **catch** methods. Add output to the console before and after calling this function. Use **Category.JavaScript** and **Category.Software** as the parameter value.	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 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.	4 Return from the function registered with **then()** the number of books found. Register with another **then()** method a function that should display the number of books found in the console.	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.
5	5	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()**	In the **types.ts** file, create an alias of type **Unpromisify<T>**, which should return the type of the promise value. 6 Get the return type of the **getBooksByCategoryPromise()** function using the **typeof** operator and the **ReturnType** utility 7 Apply **Unpromisify<T>** to the type returned by **getBooksByCategoryPromise()**	В файле **types.ts** создайте алиас типа **Unpromisify<T>**, который должен возвращать тип значения промиса. 6 Получите тип возвращаемого значения функции **getBooksByCategoryPromise(),** используя **typeof** оператор и утилиту **ReturnType** 7 Примените **Unpromisify<T>** к полученому типу, который возвращает функция **getBooksByCategoryPromise()**

Завдання 09.03. Асинхронні функції	**Task 09.03. Async functions**	**Задание 09.03. Асинхронные функции**
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. Обробіть помилку за допомогою **catch().**	1 Create an asynchronous **logSearchResults()** function in the **funtions.ts** file. The function should use the **getBooksByCategoryPromise**() function, get and output to the console the number of books found. 2 Call this function. Set **Category.JavaScript** as the value of the parameter. Add output to the console before and after the function call. Handle the error with **catch().**	1 Создайте асинхронную функцию **logSearchResults()** в файле **funtions.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().**