TypeScript Lab Assignments

Exercise 1 - Greetings!

Display a greeting message.

- Create a simple function that takes two arguments.
- The first argument is a string.
- The second argument is a number.
- Make a greeting message using both parameters and a template string.
- The function should return the greeting message.
- Alert or log the result of the function

Exercise 2 - Shape up!

Create an interface that defines the properties of a shape.

- color - string
- area - method that calculates and returns the area of the shape
- toString - method that returns the name of the shape as well as its color and area

Create at least two classes that implements the interface and has the listed private properties.

- Circle
  - radius - number
- Rectangle
  - width - number
  - height - number

Create an instance of each shape and alert or log the result of each objects toString() method.

Example output for the circle class:

```
Circle - 3.14159 - green
```

Functions

We are going to create our to-do list using functions. Let's consider what we need to implement all the necessary functions for our list.

First of all, we need to be able to save tasks. To do this, we will need somewhere to store them. We will use an array, which will be able to:

- Add a new task (a string).
- Remove a task that we've completed.
- Iterate over all the tasks so that we can see them.

Each of these features is a function that we will have to create. To add and remove tasks, we will need to specify as a parameter the string for the task itself ("Buy eggs", for example).

Once we have inserted or removed the task, we should return the number of elements that are in the list. We should also print the following in the console:

For adding a task

```
=========== NEW TASK ===========
Task "Buy eggs" inserted in the list
Number of items: 1
```

For removing a task

```
=========== TASK REMOVED ===========
Task "Buy eggs" removed from the list
Number of items: 0
```

In both cases, "Buy eggs" is the text of the task we just inserted/deleted.

Example: If the first task we add is "Start working with TypeScript", the output should be the following:

```
=========== NEW TASK ===========
Task "Start working with TypeScript" inserted in the list
Number of items: 1
```

We also need a function to list all of our tasks. For this function, we just need to print in console all the items in our array. This function doesn't need a return value.

Once you have implemented all these features, you will have the first iteration completed.

Tasks

- Create an array of strings.
- Create an `addTask` function:
  - It receives a `string` as a parameter called `task`.
  - It adds the task to the array.
  - It prints a message indicating the insertion.
  - It returns the number of elements in the list and prints that number in the console as well.
- Create a `listAllTasks` function:
  - It iterates over all the tasks in the list.
  - It prints each list item in the console.
- Create a `deleteTask` function:
  - It receives a `string` as a parameter called `task`.
  - It removes that string from the array.
  - It returns the number of elements in the list after the deletion.
  - It prints a message indicating the deletion.
  - It returns the number of elements in the list and prints that number in the console as well.

Class with one Interface

We are going to implement the same functionality by using an Interface. Remember that an interface specifies all the functions that a class will have, and also the parameters of each function.

We will create an Interface in the `todointerface.ts` file that will specify the different properties and methods described in the first iteration:

- An Array of tasks.
- A function to add tasks that will receive the task as a parameter.
- A function to list all the tasks.
- A function to delete tasks that will receive the task as a parameter.

Once we have defined and exported the Interface, we will create a class called `Todo` that will implement the interface and its methods. We are going to add this class in the `main.ts` file.
We will keep the same inputs and outputs we used in the previous iteration.

Tasks

Interface (`todointerface.ts`)

- Array of strings.
- `addTask` function:
  - It receives a `string` as a parameter
  - It returns a number.
- `listAllTasks` function:
  - It has no parameters.
  - It doesn't return anything.
- `deleteTask` function:
  - It receives a `string` as a parameter
  - It returns a number.

Class (`main.ts`)

- Create a class that implements the Interface:
- Remember that you need to import the interface before you use it.
- Implement all the properties and functions indicated in the interface:
- Create an array of strings.
- Create an `addTask` function:
  - It receives a `string` as a parameter.
  - It prints the same message as iteration 1.
  - It returns the number of elements in the list and prints this number to the console as well.
- Create a `listAllTasks` function:
  - It prints in console all the tasks of the list.
  - It doesn't return anything.
- Create a `deleteTask` function:
  - It receives a `string` as a parameter.
  - It prints the same message as iteration 1.
  - It returns the number of elements in the list and prints the number in the console also.

Two classes with two interfaces

We are going to create the same Todo list by adding another interface (and class) to our implementation.

When we have different elements in a project, it's very common (and also a good practice) to treat each of them as an independent element and create a class and

interface for each of these elements. In this case, we have been creating a Todo list without considering the possibility of splitting that into two different parts: the list, and the items.

In this iteration, we are going to create two different interfaces, one for the list elements, and another one for the list itself. So inside the list, we are going to have a reference to the other interface.

The list item interface will have the following properties:

- Title of the task, which will be a string.
- Status of the task, represented by a boolean value.
- updatedAt, which will store the date when the task was last modified.
- toggleStatus function, which will set the status as true if it's false, and vice versa.

In the Todo list interface, we will have the following properties:

- An array of list item elements.
- A function to add a task, which will receive the TodoItem as a parameter.
- A function to list all the items in the list.
- A function to delete a task, which will receive the TodoItem as a parameter.

Once we have implemented both interfaces, we will have to create the classes that will implement them. In the `main.ts` file, we will add a `TodoItem` class, which should implement the `TodoItemInterface`, and the `TodoList` class, which should implement the `TodoListInterface` interface.
Once you are done with that, the exercise will be finished!

Tasks

Interfaces (`interfaces.ts`)

- TodoItemInterface
  - Define a `title` as string.
  - Add a `status` as a boolean.
  - `updatedAt` is a Date that represents when the task was last updated.
  - Add a function called `toggleStatus` that will update the task status.
- TodoListInterface
  - Define an Array of `TodoItemS`.
  - Create an `addTask` function:
    - It receives a `TodoItem` as a parameter.

        - It returns the number of elements in the list and prints the number.
    - Add a `listAllTasks` function:
        - It doesn't return anything.
        - It lists all the titles of all the TodoItems in the list
    - Create a `deleteTask` function:
        - It receives a `TodoItem` as a parameter.
        - It returns the number of elements in the list and prints the number.

Class (`main.ts`)

- Create a `TodoItem` class that implements the `TodoItemInterface` interface, adding all the required properties and functions specified in the interface:
    - `title` string field, to indicate what the task entails.
    - `status` that will indicate if the task is completed or not.
    - `updatedAt` date field, to indicate the date that the task was last updated.
    - We have to add a `constructor` to the class to set up the task `title` and the `updatedAt` values when creating a task.
    - `toggleStatus` function to change the status of a `TodoItem`. It should:
        - Change the status of the task. If the status is currently 'finished', the new status will be 'unfinished', and viceversa.
        - Update the `updatedAt` value, by setting up the current date.

        Without supplying a parameter, how can you accomplish this with a simple function?

- Create a `TodoList` class that implements the `TodoListInterface` interface, adding all the required properties and functions specified in the interface:
    - An `Array` of `TodoItem`s to be able to save the different tasks you want to do.
    - Create an `addTask` function:
        - It receives a string as a parameter.
        - It prints in console the same message as iteration 1.
        - It returns the number of elements in the list and prints the number.
    - Create a `listAllTasks` function:
        - It prints all the tasks of the list. In this case, we want to print the title and the updated date of the task.
        - It doesn't return anything.
    - Create a `deleteTask` function:
        - It receives a string as a parameter.
        - It prints in console the same message as iteration 1.

- It returns the number of elements in the list and prints the number.

Happy coding!