Lab 2: Types, Variables and Functions

In this lab, you'll be creating a simple task manager. you'll learn how to declare types, use type annotations with variables and functions.

Working with types

In the first section, we'll create new types using interfaces, add and manipulate properties.

1. Open the starter project with VSCode
2. Transform the JS File `main.js` file in the src folder into a TS file by changing the extension (`main.ts`)
3. Open the file and create an interface `Task`

```
interface Task {

}
```

4. The type `Task` should represent the values of the variable `tasks`, add the correponding properties: `id`, `title`, `description` and `date` to the interface and add type annotations

```
    interface Task {
        id: number;
        title: string;
        description: string;
        date: Date
}
```

5. Add another property `assignedTo` of type `Person` to the `Task` interface

```
assignedTo: Person;
```

6. Declare a new interface `Person` having the properties: `id`, `firstName` and `lastName`

```
 interface Person {
     id: number;
```

```
        firstName: string;
        lastName: string;
    }
```

7. Add anther property `data` to `Task` of `any` type and make it **optional**

```
data?: any
```

8. Now, using `type` aliases , create another type `Tasks` which represent an array of `Task`

```
type Tasks = Array<Task>
// or type Tasks = Task[]
```

9. Type-check your code by running `tsc` command at the project directory

10. Add type annotation to `task` and `tasks` variables, they should be of type `Task` and `Tasks` respectivelly.

11. Run `tsc` again and see the logs.

> This time it should show an error because property `assignedTo` is missing in `task` variable and required in type `Task`, we'll fix this error in the next section.

Duck-typing (Structural typing)

Next, let's see how we can take advantage of the duck-typing when creating objects:

1. Create a new variable `me` (use `let`, `var` or `const` keyword)

2. The variable `me` should contain the two properties: `id`, `firstName` and `lastName`

```
let me = {
    id: 1,
    firstName: "Typescript",
    lastName: "Developer"
}
```

3. Add the missing proprety `assignedTo` in the `task` variable and assign to `me` variable

```
assignedTo: me
```

4. Run `tsc` command

   > Although `me` is not of type `Person` , but Typescript compiler accept it because the value of me and the type Person have the same structure, this is the duck-typing technique used by Typescript

5. We want to inform typescript compiler that `me` value if of type Person, we can do that using type assertion like the following:

   ```
   assignedTo: me as Person //or <Person> me
   ```

6. Run `tsc` again.

Functions

Typescript supports type annotations for function's parameters types and return type:

1. In the same file (`data.ts`), create a function `addTask` that should takes all the properties of the type `Task` and add type annotations for the parameters and return type (it should return a `Task` object)

   ```
   function addTask(id: number, title: string, description: string, date: Date,
                                       assignedTo: Person, data?: any): Task {
       // TODO implement
   }
   ```

2. Add a **default** values for `assignedTo` and `data` properties

   ```
   (..., assignedTo: Person = me, data: any = {})
   ```

3. Create a second function `updateTask` that have the same signature of `addTask` function

4. Create another function deleteTask which take a single parameter `id` and return a `boolean` (true if the task is deleted, false otherwise)

   ```
   function deleteTask(id: number): boolean {
       // TODO implement
   }
   ```

5. Implement all of the functions above (`addTask` , `updateTask` , `deleteTask`)
6. Using **arrow-function** syntax, create a function `printTasks` that takes no parameter and return nothing (`void`).

   ```
   let printTasks = () : void => {
       //TODO implement
   }
   ```

   Implement the function so it logs all the tasks (`tasks` variable) following this format :

   ```
   Task 1: Fixing bugs, assigned to : Typescript Developer, due date : 2022-12-12
   ```

7. Call `printTasks()` in the end of file
8. Run `tsc` to type-check
9. Run the following command to run the output js file

   ```
   node  dist/main.js
   ```