100 XP

Lab - Use functions in TypeScript

20 minutes

In this lab, you'll convert some JavaScript functions to TypeScript.

Exercise 1: Convert a JavaScript function to TypeScript with strongly typed parameters and return types

The following JavaScript code contains three functions:

- `BuildArray` builds an array of unique random numbers. It accepts an `items` parameter that determines the number of items in the array and a `sortOrder` parameter that determines whether the array is sorted in ascending or descending order.
- `sortDescending` and `sortAscending` are comparison functions that tell the `sort()` method how to sort numbers in ascending or descending order.

Convert the functions to TypeScript with strongly typed parameters and return types.

1. Clone the starting repository by entering the following at the command prompt.

```
Bash

git clone https://github.com/MicrosoftDocs/mslearn-typescript
cd mslearn-typescript/code/module-04/m04-start
code .
```

2. Open the file **module04.ts**.

3. Locate `TODO: Update the BuildArray function.`

4. In the `BuildArray` function, add types to the parameter list, return value, and variables.

```
TypeScript

function buildArray(items: number, sortOrder: 'ascending' | 'descending'):
number[] {
   let randomNumbers: number[] = [];
```

```
    let nextNumber: number;

    for (let counter = 0; counter < items; counter++) {
        nextNumber = Math.ceil(Math.random() * (100 - 1));
        if (randomNumbers.indexOf(nextNumber) === -1) {
          randomNumbers.push(nextNumber);
        } else {
          counter--;
        }
    }

    if (sortOrder === 'ascending') {
      return randomNumbers.sort(sortAscending);
    } else {
      return randomNumbers.sort(sortDescending);
    }
}
```

5. Locate `TODO: Convert the sortDescending and sortAscending functions to arrow functions.`

6. Convert the `sortDescending` and `sortAscending` functions to anonymous functions and assign them to variables of the same name.

TypeScript

```
let sortDescending = (a, b) => {
    if (a > b) {
        return -1;
    } else if (b > a) {
        return 1;
    } else {
        return 0;
    }
}

let sortAscending = (a, b) => {
     if (a > b) {
       return 1;
     } else if (b > a) {
       return -1;
     } else {
       return 0;
     }
   }
```

7. Locate `TODO: Declare a new function type for the sortDescending and sortAscending functions.`

8. Declare a new function type for the `sortDescending` and `sortAscending` functions using either a type alias or an interface.

TypeScript

```
type compareFunctionType = (a: number, b:number) => number;
```

9. In the variable declarations for `sortDescending` and `sortAscending`, apply the new function type as the variable type.

TypeScript

```
let sortDescending: compareFunctionType = (a, b) => {
   if (a > b) {
       return -1;
   } else if (b > a) {
       return 1;
   } else {
       return 0;
   }
}

let sortAscending: compareFunctionType = (a, b) => {
    if (a > b) {
      return 1;
    } else if (b > a) {
      return -1;
    } else {
      return 0;
    }
  }
```

10. Test your work by calling the `buildArray` function.

TypeScript

```
let myArray1 = buildArray(12, 'ascending');
let myArray2 = buildArray(8, 'descending');
console.log(myArray1);
console.log(myArray2);
```

Exercise 2: Convert a JavaScript function to a TypeScript using optional parameters

This JavaScript function returns the payment amount for a loan.

1. Locate `TODO: Update the LoanCalculator function.`
2. Convert the `loanCalculator` function to TypeScript with strongly typed parameters, variables, and return types.
3. Make the `months` parameter optional but assign it a default value of `12` months if omitted.

TypeScript

```
function loanCalculator (principal: number, interestRate: number, months =
12): string {
    let interest: number = interestRate / 1200;   // Calculates the monthly
interest rate
    let payment: number;
    payment = principal * interest / (1 - (Math.pow(1/(1 + interest),
months)));
    return payment.toFixed(2);
}
```

4. Test your work by calling the `loanCalculator` function with only the required parameters.

TypeScript

```
let myLoan = loanCalculator(1000, 5);
console.log(myLoan);
```

Challenge yourself!

For an added challenge, take some existing JavaScript that you may have written or that you find on the web and rewrite it in TypeScript using what you've learned about functions. You can copy and paste the JavaScript into the Playground and edit it or use another editor like Visual Studio Code.

Lab solution

View the final version of the code by entering the following at the command prompt.

Bash

```
cd ../m04-end
code .
```

Open the file **module04.ts** to see the solution to this lab.

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆