IS 312 Web Design: TypeScript (TS) for Modern Web Application
HOP05: Functions

8/28/2019, Developed by Kevin Kuanting Chen, Class of 2020
8/20/2020, Revised by Amrutha Vaidyanathan, Class of 2020
9/23/2020, Revised by Kim Nguyen, Class of 2021

School of Technology & Computing (STC)
City University of Seattle (CityU)

Before You Start

- The directory path shown in screenshots may be different from yours.
- Some steps are not explained in the tutorial**.** If you are not sure what to do:
    1. Consult the resources listed below.
    2. If you cannot solve the problem after a few tries, ask a TA for help.

Learning Outcomes

Students will be able to:

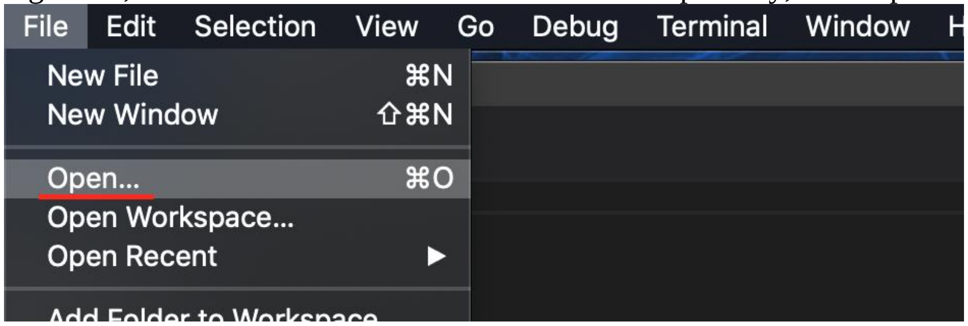
- Understand different functions in TypeScript
- Understand different ways to pass parameters in TypeScript functions

Resources

- W3School Data Types

Preparation

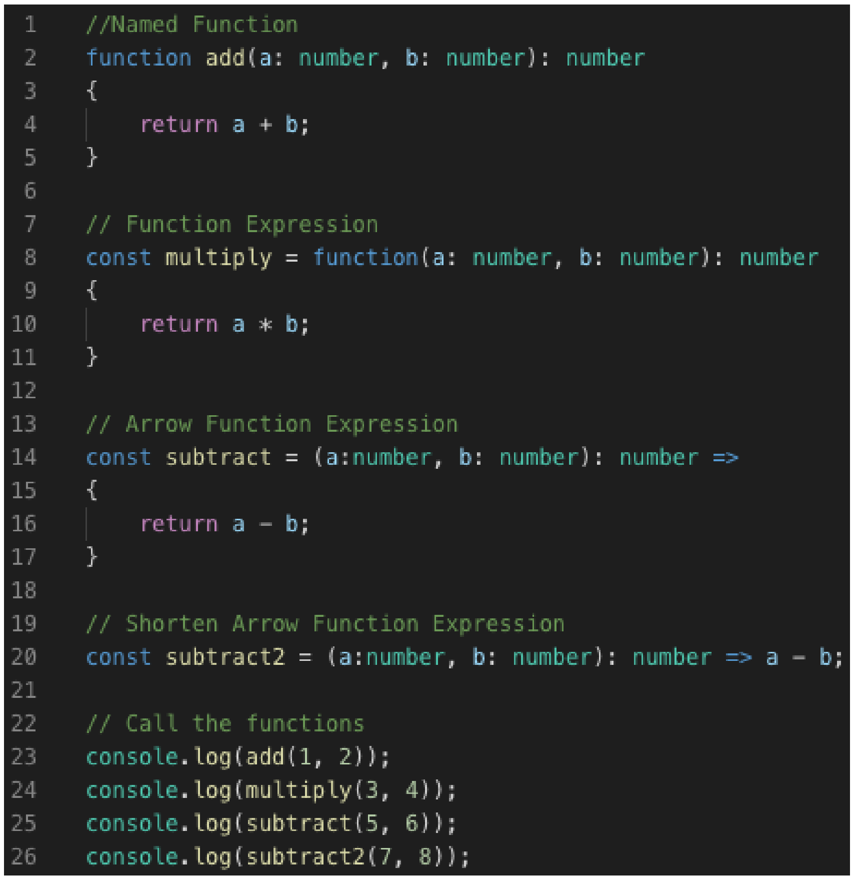
1. Open the VS Code and open the repository you cloned from Github, if you have not cloned, go back to the Github repository generated when you accept the HOP assignment, read the instruction on how to clone the repository, before proceeding.



2. Open the terminal from the VSCode, check your current directory using the following command, if you are in **Module 5**, you are in the right place:

Functions

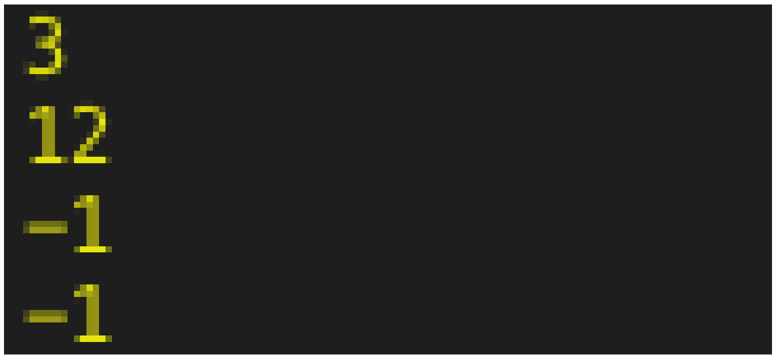
1) Create a **functions.ts** file under this module's folder and type the following code in the file.



```
//Named Function
function add(a: number, b: number): number
{
    return a + b;
}

// Function Expression
const multiply = function(a: number, b: number): number
{
    return a * b;
}

// Arrow Function Expression
const subtract = (a:number, b: number): number =>
{
    return a - b;
}

// Shorten Arrow Function Expression
const subtract2 = (a:number, b: number): number => a - b;

// Call the functions
console.log(add(1, 2));
console.log(multiply(3, 4));
console.log(subtract(5, 6));
console.log(subtract2(7, 8));
```

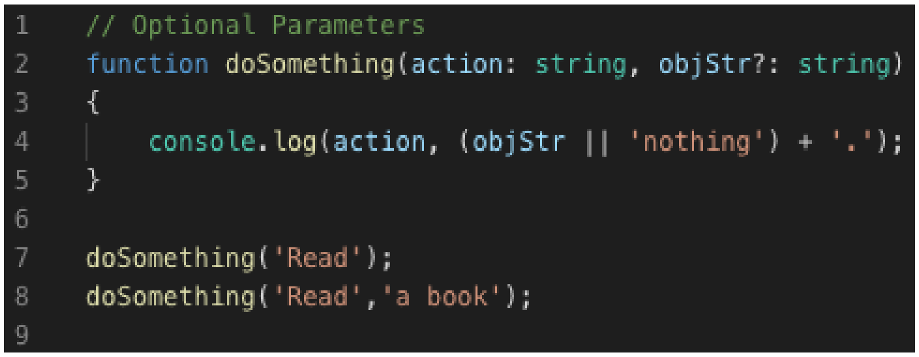
2) Compile your TypeScript code and run the JavaScript file.

   >>> tsc functions.ts
   >>> node functions.js



```
3
12
-1
-1
```

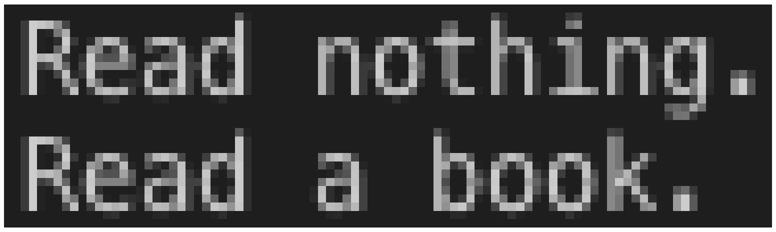
3) Create another file called **more_functions.ts** in the folder and type the following code into the file.



```typescript
// Optional Parameters
function doSomething(action: string, objStr?: string)
{
    console.log(action, (objStr || 'nothing') + '.');
}

doSomething('Read');
doSomething('Read','a book');

```

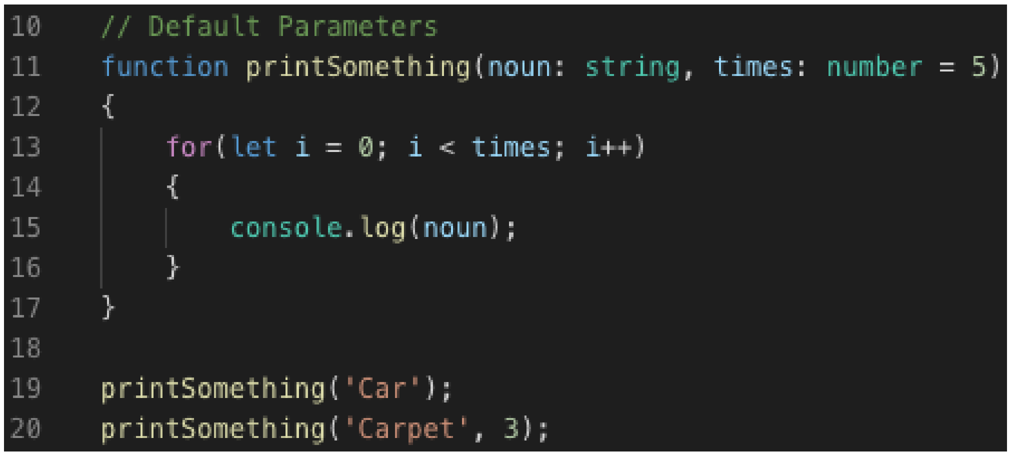
4) Compile your TypeScript code and run the JavaScript file.

   >>> tsc more_functions.ts
   >>> node more_functions.js



```
Read nothing.
Read a book.
```

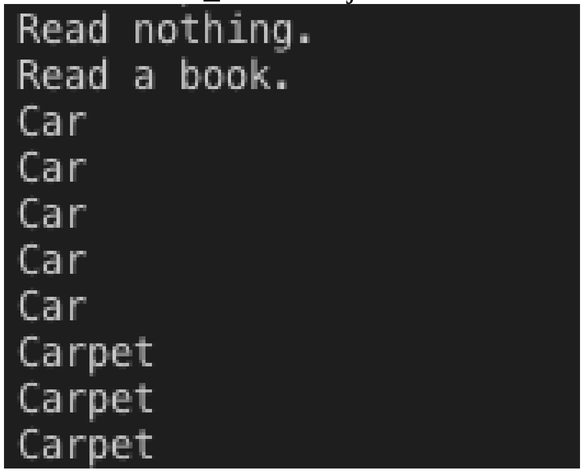
5) Add the following code into the **more_functions.ts** file.



```
// Default Parameters
function printSomething(noun: string, times: number = 5)
{
    for(let i = 0; i < times; i++)
    {
        console.log(noun);
    }
}

printSomething('Car');
printSomething('Carpet', 3);
```

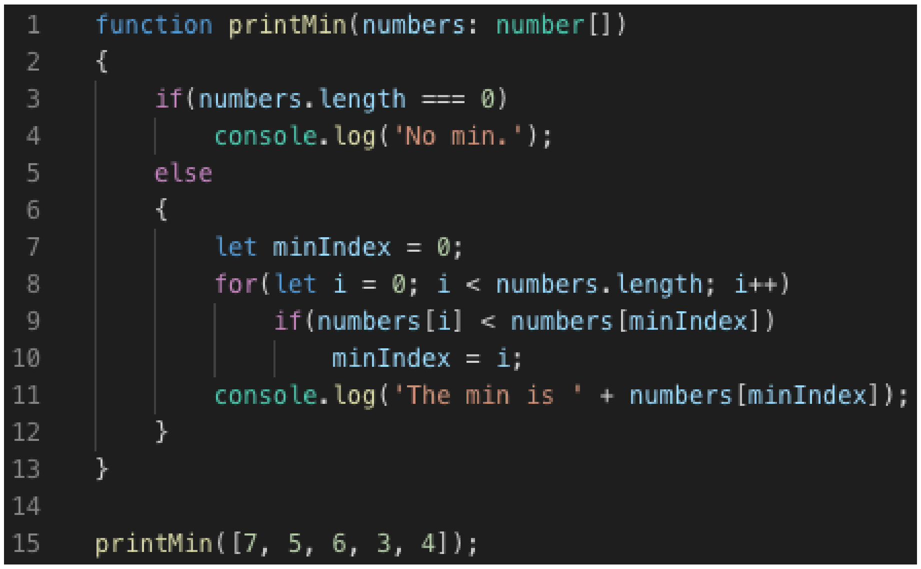
6) Compile your TypeScript code and run the JavaScript file again.

   >>> tsc more_functions.ts & node more_functions.js



```
Read nothing.
Read a book.
Car
Car
Car
Car
Car
Carpet
Carpet
Carpet
```

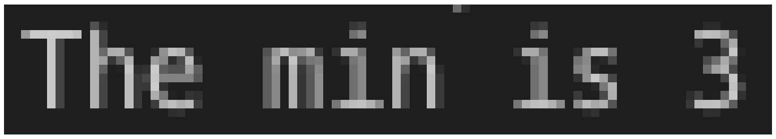
7) Create another file called **min.ts** in the folder and type the following code in the file.



```
function printMin(numbers: number[])
{
    if(numbers.length === 0)
        console.log('No min.');
    else
    {
        let minIndex = 0;
        for(let i = 0; i < numbers.length; i++)
            if(numbers[i] < numbers[minIndex])
                minIndex = i;
        console.log('The min is ' + numbers[minIndex]);
    }
}

printMin([7, 5, 6, 3, 4]);
```

8) Compile your TypeScript code and run the JavaScript file again.

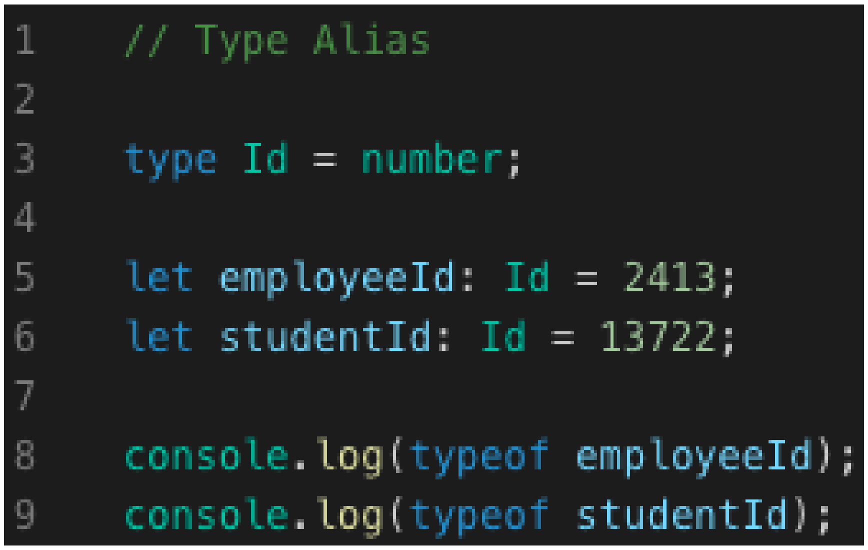
```
>>> tsc min.ts
>>> node min.js
```



```
The min is 3
```

Type Alias

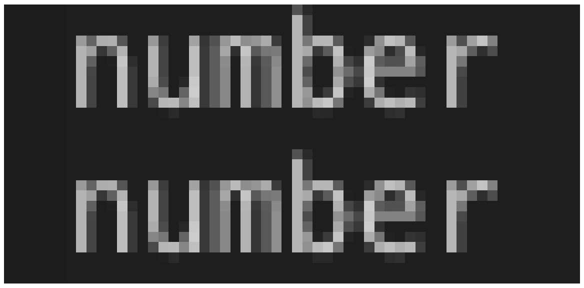
1) Create a **more_types.ts** file under this module's folder and type the following code in the file.



```
// Type Alias

type Id = number;

let employeeId: Id = 2413;
let studentId: Id = 13722;

console.log(typeof employeeId);
console.log(typeof studentId);
```

2) Compile your TypeScript code and run the JavaScript file again.

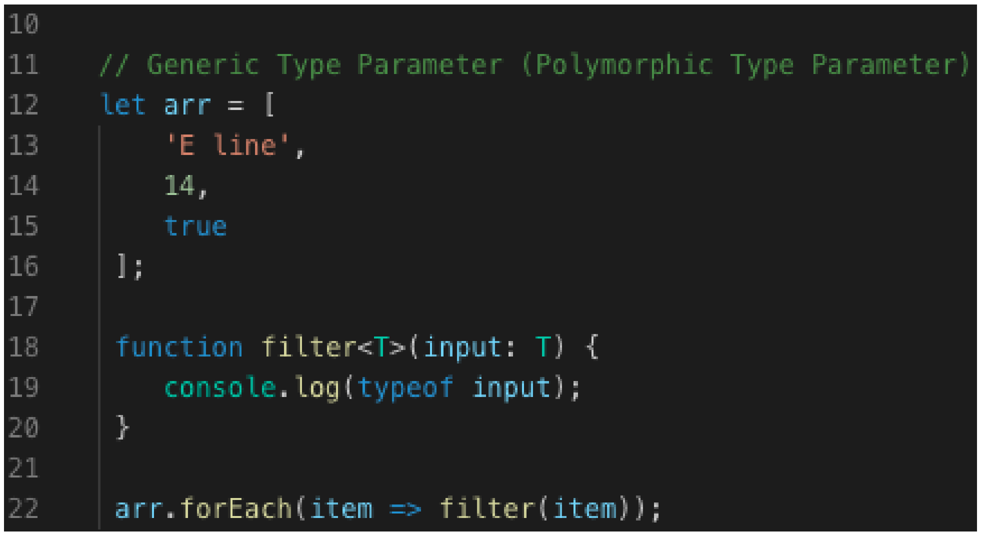
```
>>> tsc more_types.ts
>>> node more_types.js
```



```
number
number
```

Generic Type Parameter (Polymorphic Type Parameter)

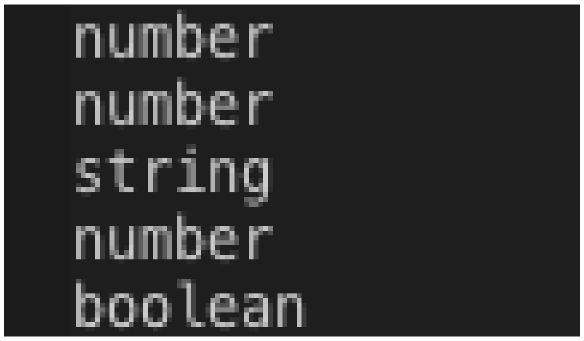
1) Add the following code into the **more_types.ts** file.



```
// Generic Type Parameter (Polymorphic Type Parameter)
let arr = [
    'E line',
    14,
    true
 ];

 function filter<T>(input: T) {
    console.log(typeof input);
 }

 arr.forEach(item => filter(item));
```

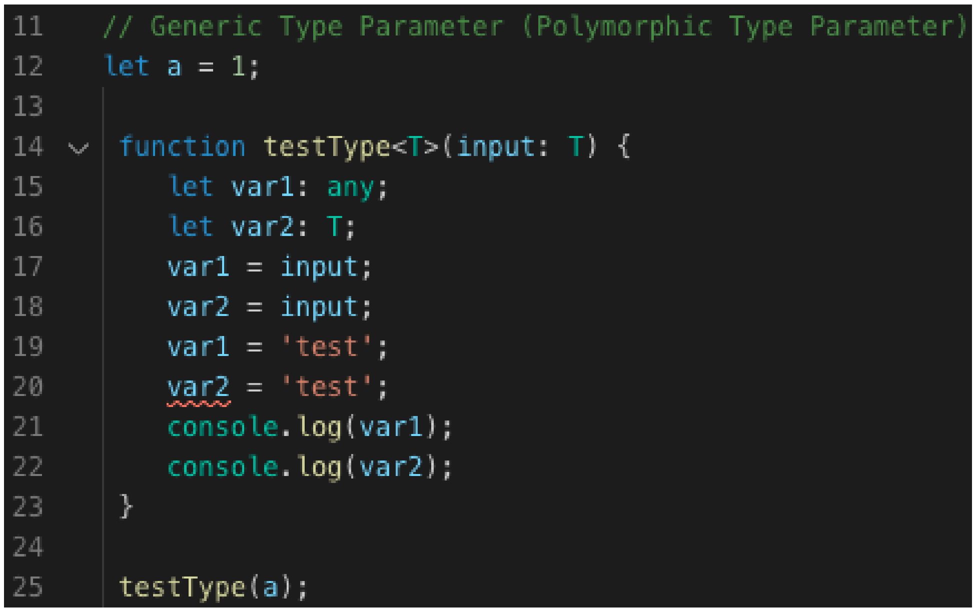
2) Compile your TypeScript code and run the JavaScript file again.

   >>> tsc more_types.ts & node more_types.js



```
number
number
string
number
boolean
```

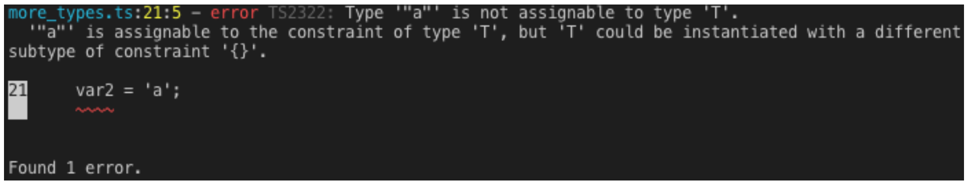
3) Update the file to match the following picture.



```
// Generic Type Parameter (Polymorphic Type Parameter)
let a = 1;

function testType<T>(input: T) {
    let var1: any;
    let var2: T;
    var1 = input;
    var2 = input;
    var1 = 'test';
    var2 = 'test';
    console.log(var1);
    console.log(var2);
}

testType(a);
```

4) Compile your TypeScript code. **You will get an error message.** Read the error message and think about why you get it. (Don't forget you can easily get hints by moving your mouse to the error in VSCode)

   >>> tsc more_types.ts

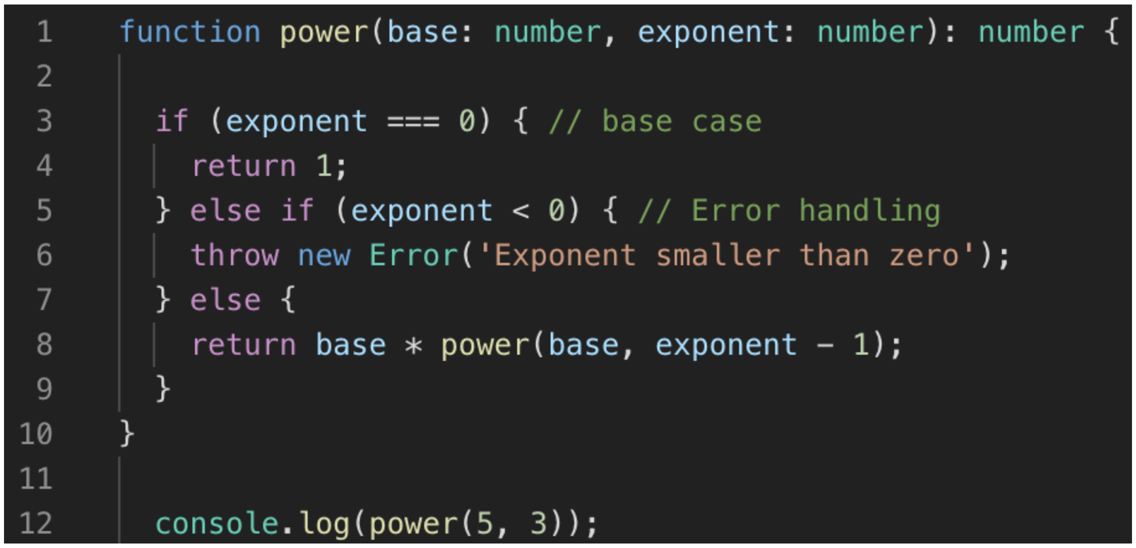


```
more_types.ts:21:5 - error TS2322: Type '"a"' is not assignable to type 'T'.
  '"a"' is assignable to the constraint of type 'T', but 'T' could be instantiated with a different
subtype of constraint '{}'.

21     var2 = 'a';
       ~~~~


Found 1 error.
```

Explanations
A generic type parameter, or a polymorphic type parameter, not only enables variables to take in any data type just like the keyword *any*, but also helps the compiler keep your code type-safe in case you accidentally change the variables' data type.

Recursion

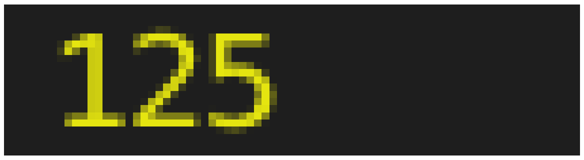
1) You can call a function inside itself. Create a **recursion.ts** file under this module's folder.



```
function power(base: number, exponent: number): number {

  if (exponent === 0) { // base case
    return 1;
  } else if (exponent < 0) { // Error handling
    throw new Error('Exponent smaller than zero');
  } else {
    return base * power(base, exponent - 1);
  }
}

  console.log(power(5, 3));
```

Note: Just beware of the base case (you want it to stop at some point) and stack overflows when using recursions.

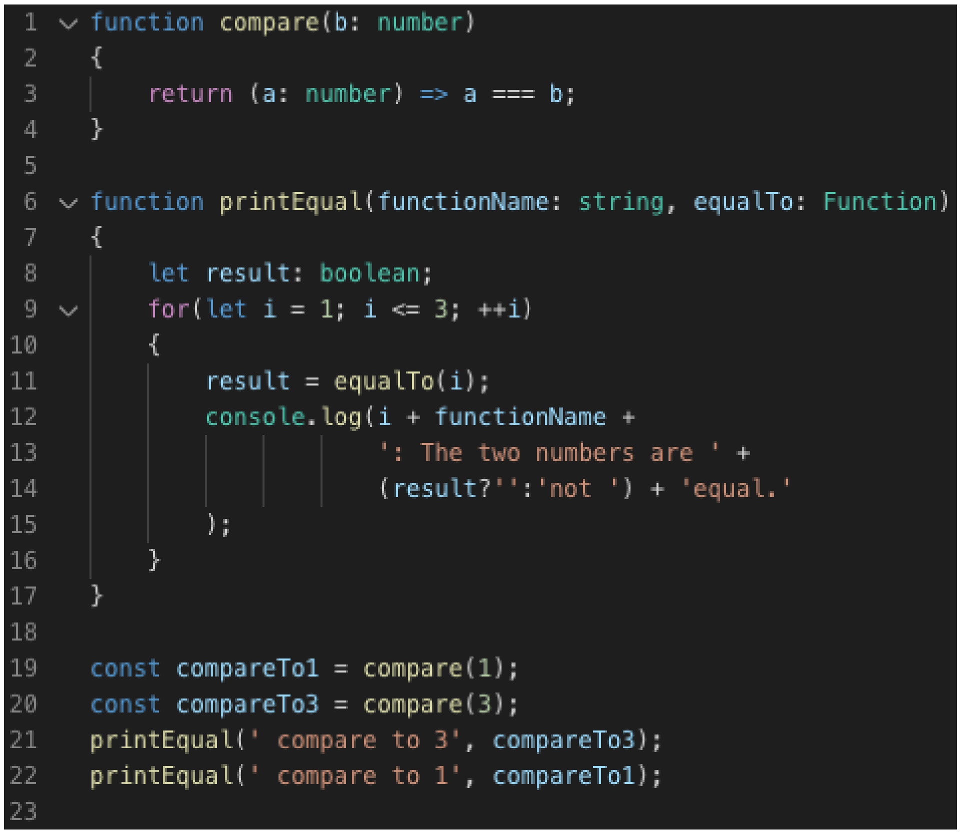
2) Compile your TypeScript code and run the JavaScript file.



```
125
```

Higher-Order Functions

1) Functions are objects in JavaScript and TypeScript. This means you can pass them as arguments or return them just like other variables.
   Create a **higher_order.ts** file under this module's folder and type the following code in the file.



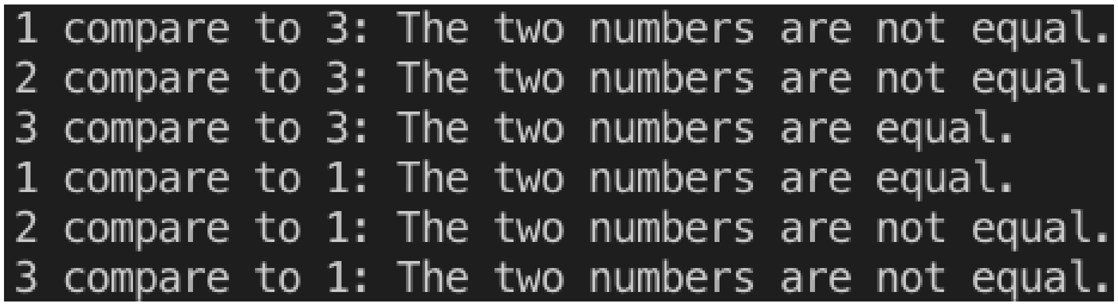
```
function compare(b: number)
{
    return (a: number) => a === b;
}

function printEqual(functionName: string, equalTo: Function)
{
    let result: boolean;
    for(let i = 1; i <= 3; ++i)
    {
        result = equalTo(i);
        console.log(i + functionName +
                    ': The two numbers are ' +
                    (result?'':'not ') + 'equal.'
        );
    }
}

const compareTo1 = compare(1);
const compareTo3 = compare(3);
printEqual(' compare to 3', compareTo3);
printEqual(' compare to 1', compareTo1);
```

   Note: If **equalTo** is showing as error change it is **equalto**

   Note: The "?:" in "(BOOLEAN TO TEST) ? (DO IF TRUE) : (DO IF FALSE)" is called a ternary operator which is mentioned in the first chapter in the textbook (https://eloquentjavascript.net/01_values.html). It is basically just a short way to write the if-else statement.

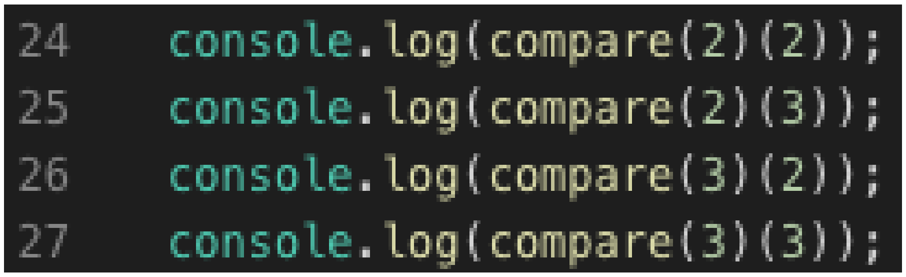
2) Compile your TypeScript code and run the JavaScript file.

```
>>> tsc higher_order.ts
>>> node higher_order.js
```



```
1 compare to 3: The two numbers are not equal.
2 compare to 3: The two numbers are not equal.
3 compare to 3: The two numbers are equal.
1 compare to 1: The two numbers are equal.
2 compare to 1: The two numbers are not equal.
3 compare to 1: The two numbers are not equal.
```

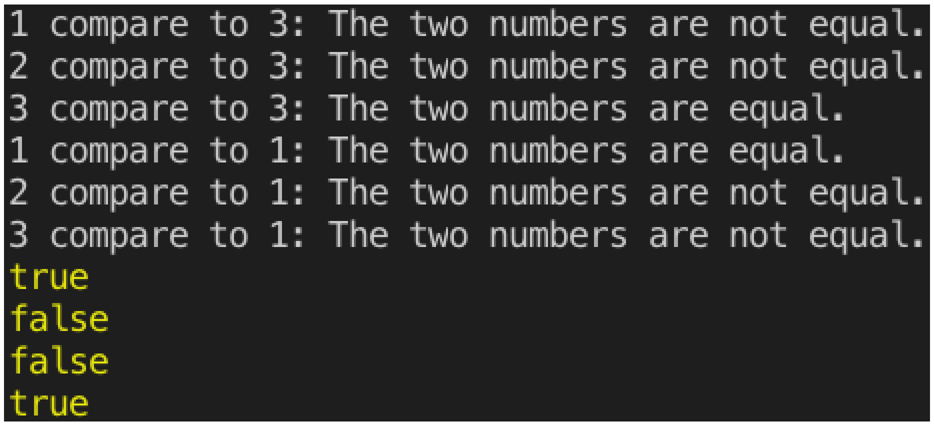
3) Add the following lines to your TypeScript file.



```
console.log(compare(2)(2));
console.log(compare(2)(3));
console.log(compare(3)(2));
console.log(compare(3)(3));
```

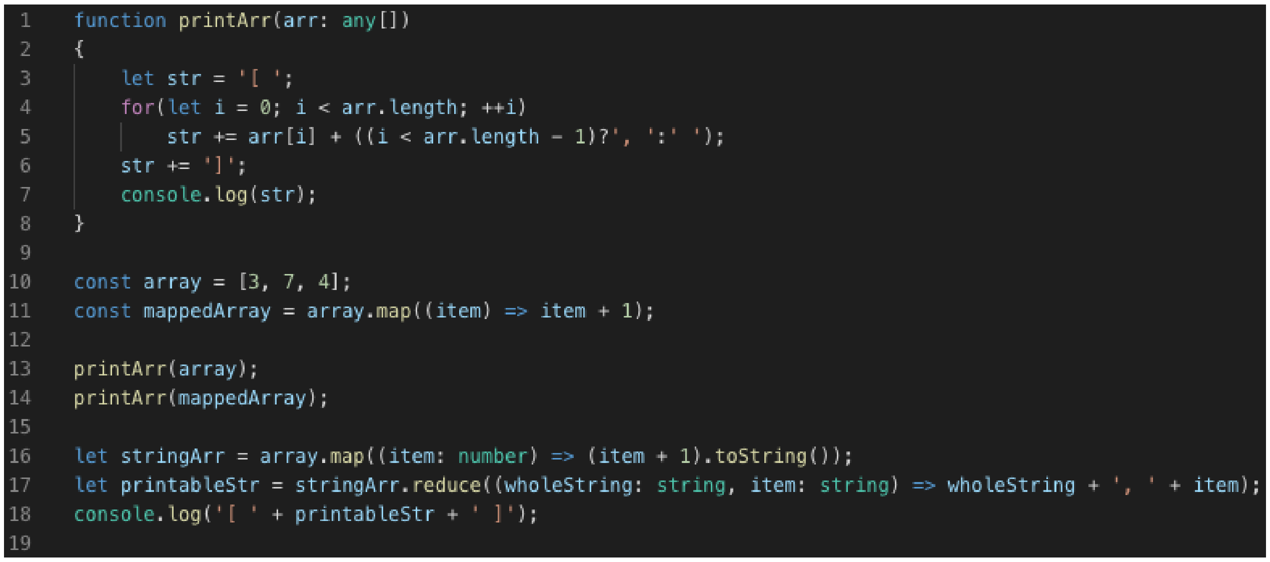
4) Compile your TypeScript code and run the JavaScript file.

   >>> tsc higher_order.ts & node higher_order.js



```
1 compare to 3: The two numbers are not equal.
2 compare to 3: The two numbers are not equal.
3 compare to 3: The two numbers are equal.
1 compare to 1: The two numbers are equal.
2 compare to 1: The two numbers are not equal.
3 compare to 1: The two numbers are not equal.
true
false
false
true
```

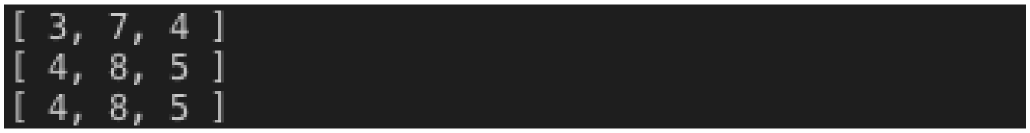
5) You may find yourself using higher-order functions when there is a function that you want to reuse many times, for instance, when performing actions on an array.
   Create an **array_methods.ts** file under this module's folder and type the following code in the file.



```
function printArr(arr: any[])
{
    let str = '[ ';
    for(let i = 0; i < arr.length; ++i)
        str += arr[i] + ((i < arr.length - 1)?', ':' ');
    str += ']';
    console.log(str);
}

const array = [3, 7, 4];
const mappedArray = array.map((item) => item + 1);

printArr(array);
printArr(mappedArray);

let stringArr = array.map((item: number) => (item + 1).toString());
let printableStr = stringArr.reduce((wholeString: string, item: string) => wholeString + ', ' + item);
console.log('[ ' + printableStr + ' ]');
```

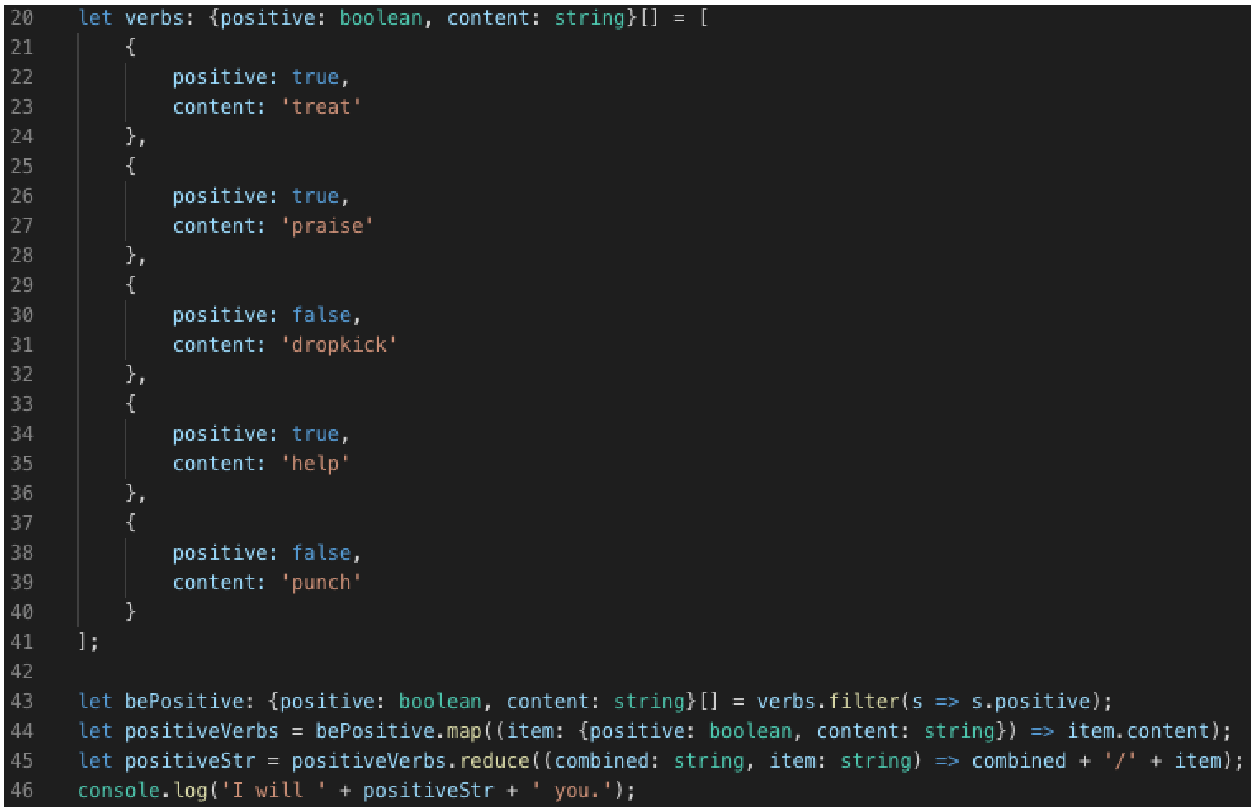
6) Compile your TypeScript code and run the JavaScript file.

```
>>> tsc array_methods.ts
>>> node array_methods.js
```



```
[ 3, 7, 4 ]
[ 4, 8, 5 ]
[ 4, 8, 5 ]
```

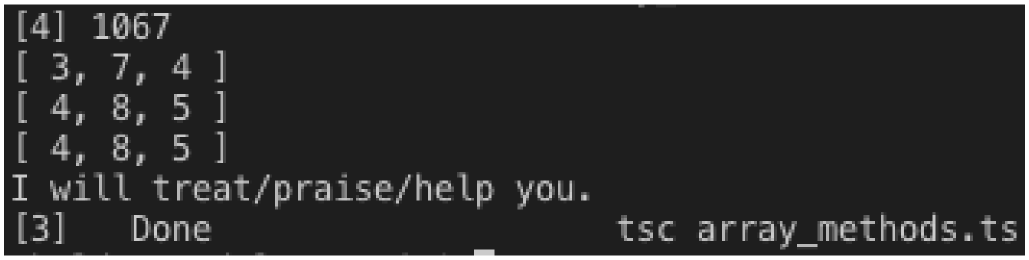
7) Add the following code to the **array_methods.ts** file.



```
let verbs: {positive: boolean, content: string}[] = [
    {
        positive: true,
        content: 'treat'
    },
    {
        positive: true,
        content: 'praise'
    },
    {
        positive: false,
        content: 'dropkick'
    },
    {
        positive: true,
        content: 'help'
    },
    {
        positive: false,
        content: 'punch'
    }
];

let bePositive: {positive: boolean, content: string}[] = verbs.filter(s => s.positive);
let positiveVerbs = bePositive.map((item: {positive: boolean, content: string}) => item.content);
let positiveStr = positiveVerbs.reduce((combined: string, item: string) => combined + '/' + item);
console.log('I will ' + positiveStr + ' you.');
```

8) Compile your TypeScript code and run the JavaScript file.

>>> tsc array_methods.ts & node array_methods.js



```
[4] 1067
[ 3, 7, 4 ]
[ 4, 8, 5 ]
[ 4, 8, 5 ]
I will treat/praise/help you.
[3]   Done                    tsc array_methods.ts
```

Push your work to GitHub

Run the following commands to push your work to the GitHub repository:
Open the terminal from the VSCode by hitting the control + ~ key and type the following command:

```
>>> git add .
>>> git commit -m “Submission for Module 5– YOUR NAME”
>>> git push origin master
```