Agenda: About Functions

- Optional and Default Parameters
- Rest Parameters
- Function Overloading
- Function Types

Optional and Default Parameters

- "?" is used for optional parameters.
- "?" **should not** be used if parameter has default value.

```
function add1(x: number, y: number, z?: number): number {
   if (!z)
      z = 0;
   return x + y + z;
};
function add2(x: number, y: number, z: number = 0): number {
   return x + y + z;
};
alert(add1(10, 20))
alert(add2(10, 20))
```

Note: Unlike plain optional parameters, default-initialized parameters don't *need* to occur after required parameters.

If a default-initialized parameter comes before a required parameter, users need to **explicitly pass undefined** to get the default initialized value.

```
function add2(x: number = 0, y: number, z: number): number {
   return x + y + z;
};
alert(add1(undefined, 10, 20))
```

Rest Parameters

Sometimes, you want to work with multiple parameters as a group, or you may not know how many parameters a function will ultimately take. In JavaScript, you can work with the arguments directly using the arguments variable that is visible inside every function body.

In TypeScript, you can gather these arguments together into a variable:

```
function add(x: number, ...numbers: number[]): number {
   for (var i = 0; i < numbers.length; i++)
      x += numbers[i];
   return x;
```

```
};
alert(10)
alert(add(10, 20, 30))
alert(add(10, 20, 30, 40))
alert(add(10, 20, 30, 40, 50))
```

Function Overloading

We need to supply multiple function types for the same function as a list of overloads.

This list is what the compiler will use to resolve function calls.

Let's create a list of overloads that describe what our "**add**" function accepts and what it ret

```
//Same function List
function add(x: string, y: string): string;
function add(x: number, y: number): number; //Function Implementation
function add(x, y): any {
   if (typeof x == "number")
      return x + y;
   else
      return x + " " + y;
};
alert(add(10, 20))     //valid
alert(add("A", "B"))   //valid
//alert(add(true, "B")) //invalid
```

Note that the function add(x,y): any piece is not part of the overload list, so it only has two overloads: one that takes two numbers and one that takes two strings. Calling add with any other parameter types would cause an error.

Function Types

While writing a function, we can add types to each of the parameters and then to the function itself which is the return type.

TypeScript can figure the return type out by looking at the return statements, so we can also optionally leave this off in many cases.

A function's type has the same two parts:

1. The type of the arguments.
2. The return type.

```
let add1: (x: number, y: number) => number
```

```
add1 =  function (x: number, y: number): number {
      return x + y;
   };
//OR the compiler can infer the types while writing the function
function add(x, y) {
   return x + y;
};
let add2: (x: number, y: number) => number
add2 = add;
alert(add1(10, 20))
alert(add2(10, 20))
```