Creating and Calling Functions

Brice Wilson

Overview

- Adding type annotations to functions
- Using optional, default, and rest parameters
- Creating arrow functions
- Overloading functions
- Declaring function types

Adding Type Annotations to Functions

```
function dullFunc(value1, value2) {
    return "I'm boring and difficult. Don't be like me.";
}
```

Implicitly assigned the type “any”

```
function funFunc(score: number, message: string): string {
    return "I've got personality and I'm helpful. Be like me!";
}
```

Using the *--noImplicitAny* Compiler Option

```
function dullFunc(value1, value2) {
    return "I'm boring and difficult. Don't be like me.";
}
```

```
error TS7006: Parameter 'value1' implicitly has an 'any' type.
error TS7006: Parameter 'value2' implicitly has an 'any' type.
```

```
function CreateCustomer(name: string, age?: number) { }

function GetBookByTitle(title: string = 'The C Programming Language') { }

function GetBookByTitle(title: string = GetMostPopularBook()) { }
```

Optional and Default Parameters

Optional parameters denoted with “?” after parameter name

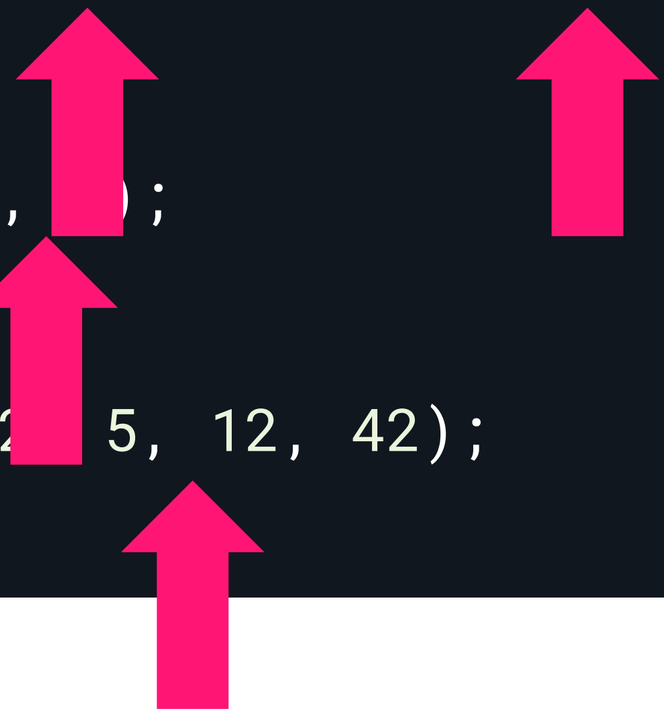
Must appear after all required parameters

Default parameters may be set to a literal value or an expression

```
function GetBooksReadForCust(name: string, ...bookIDs: number[]) { }

let books = GetBooksReadForCust('Leigh', 2, );

let books = GetBooksReadForCust('Daniel', 2, 5, 12, 42);
```



Rest Parameters

Collects a group of parameters into a single array

Denoted with an ellipsis prefix on last parameter

Demo

Annotating functions and using parameters

Anatomy of an Arrow Function

```
parameters => function body
```

Anatomy of an Arrow Function

```
let squareit = x => x * x;
let result = squareit(4); // 16


let adder = (a, b) => a + b;
let sum = adder(2, 3); // 5


let greeting = () => console.log('Hello World!');
greeting(); // Hello World!
```

Anatomy of an Arrow Function

```
let scores: number[] = [70, 125, 85, 110];
let highScores: number[];
highScores = scores.filter((element, index, array) => {
    if (element > 100) {
        return true;
    }
});
```

Demo

Converting a traditional function to an arrow function

Demo

Converting a traditional function to an arrow function...with type annotations!

Function Overloads

One symbol name

Multiple function types

One implementation with type guards

Implementing Function Overloads

```
function GetTitles(author: string): string[];
function GetTitles(available: boolean): string[];
function GetTitles(bookProperty: any): string[] {
    if(typeof bookProperty == 'string') {
        // get books by author, add to foundTitles
    }
    else if(typeof bookProperty == 'boolean') {
        // get books by availability, add to foundTitles
    }
    return foundTitles;
}
```

Demo

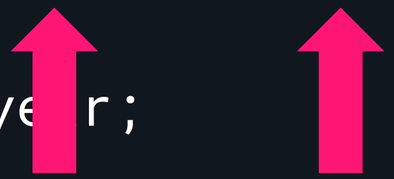
Overloading functions

```
function ReleaseMessage(year: number): string {
    return 'Year released: ' + year;
}

let releaseFunc: (someYear: number) => string;

releaseFunc = ReleaseMessage;

let message: string = releaseFunc(2024);
```



Function Types

Combination of parameter types and return type

Variables may be declared with function types

Function assigned must have the same signature as the variable type

Demo

Declaring and using function types

Summary

JavaScript functions with more features!

Type annotations

Parameters

Arrow functions

Overloads

Function types

Up Next:

Working with Interfaces