

Writing Better Functions with TypeScript



Brice Wilson

@brice_wilson www.BriceWilson.net



Overview



Adding type annotations to functions

Using arrow functions

Declaring function types



Adding Type Annotations to Functions

```
function dullFunc(value1, value2) {  
  
}
```



Adding Type Annotations to Functions

```
function dullFunc(value1, value2) {  
    }  
}
```

Implicitly assigned the type "any"



Adding Type Annotations to Functions

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



Adding Type Annotations to Functions

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


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



Adding Type Annotations to Functions

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


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



Adding Type Annotations to Functions

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


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



Adding Type Annotations to Functions

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

```
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.";  
}
```



Using the *--noImplicitAny* Compiler Option



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




Default-Initialized Parameters

```
function sendGreeting(greeting: string = 'Good morning!'): void {  
      
}
```



Default-Initialized Parameters

```
function sendGreeting(greeting: string = 'Good morning!'): void {  
  
}
```




Default-Initialized Parameters

```
function sendGreeting(greeting: string = 'Good morning!'): void {  
    console.log(greeting);  
}
```



Default-Initialized Parameters

```
function sendGreeting(greeting: string = 'Good morning!'): void {  
    console.log(greeting);  
}
```



Default-Initialized Parameters

```
function sendGreeting(greeting: string = 'Good morning!'): void {  
    console.log(greeting);  
}
```

```
sendGreeting(); // Good morning!
```

```
sendGreeting('Good afternoon!'); // Good afternoon!
```



Default-Initialized Parameters

```
function sendGreeting(greeting: string = 'Good morning!'): void {  
    console.log(greeting);  
}
```

sendGreeting(); // Good morning! 

sendGreeting('Good afternoon!'); // Good afternoon! 



Demo



Adding type annotations to parameters
and return values



Demo




Adding type annotations and default parameter values



Anatomy of an Arrow Function

parameters \Rightarrow function body



Anatomy of an Arrow Function

```
let squareIt = x => x * x;
```



Anatomy of an Arrow Function

```
let squareIt = x => x * x;
```



Anatomy of an Arrow Function

```
let squareIt = x => x * x;
```



Anatomy of an Arrow Function

```
let squareIt = x => x * x;
```



Anatomy of an Arrow Function

```
let squareIt = x => x * x;
```



Anatomy of an Arrow Function

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



Anatomy of an Arrow Function

```
let squareit = x => x * x;
```

```
let result = squareit(4); // 16
```

```
let adder = (a, b) => a + b;
```



Anatomy of an Arrow Function

```
let squareit = x => x * x;
```

```
let result = squareit(4); // 16
```

```
let adder = (a, b) => a + b;
```



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
```



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!');
```



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!');
```



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;  
    }  
});
```



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;  
    }  
});
```




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;  
    }  
});
```



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...with type annotations!



Demo



Taking advantage of function types



Summary



TypeScript functions are easier to use

Flexibility included

Clean syntax



Up Next:

Creating and Using Custom Types

