Functions

Introduction

Functions in TypeScript closely resembles JavaScript

```
const z = 100;
function sumWithZ(x, y) {
  return x + y + z;
}
```

Typed functions

Create a strongly typed function using type annotations.

```
const z = 100;
function sumWithZ(x: number, y: number): number {
    return x + y + z;
}

// Alternative
const sumWithZCopy = (x: number, y: number) => {
    return x + y + z;
}
```

Expect errors when you are doing something wrong.

```
const z = 'Hello world!'

function sumWithZ(x: number, y: number): number {
    return x + y + z;
    // => Type 'string' is not assignable to type 'number'.
}
```

Optional parameters

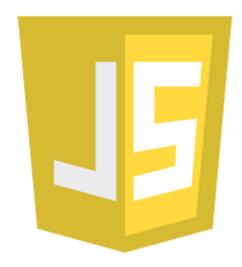
Optional parameters are supported using?

```
function printName(firstName: string, lastName?: string) {
   if (lastName) {
      console.log(`${firstName} ${lastName}`);
   } else {
      console.log(firstName);
   }
}
```

Optional parameters must be placed after required parameters.

```
function logToConsole(firstName?: string, lastName: string) {
   // => A required parameter cannot follow an optional parameter.
}
```

Default parameters



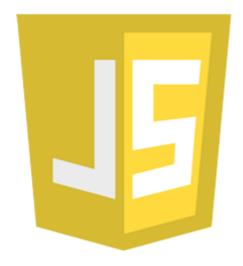
You can now use default parameters using =

```
// TypeScript
function hello(message = 'world') {
   console.log('Hello ' + message);
}
```

```
// Old fashion ES5
function hello(message) {
   if (message === undefined) {
      message = 'world';
   }
   console.log('Hello ' + message);
}
```

Default parameters should be placed after required parameters

Rest parameters



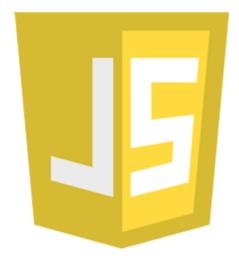
```
//TypeScript
function buyNewCar(brand: string,
    ...features: string[]) {
    console.log(`${brand}
        with features: ${features.join(', ')}`);
}

buyNewCar('Tesla',
    'Leather seats', 'Blue painting');

// => "Tesla with features:
    Leather seats, Blue painting"
```

Rest parameters must be placed after required parameters

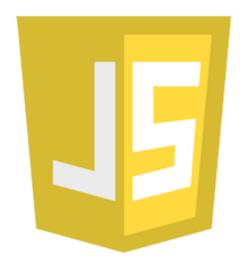
Spread operator



```
function max(numbers: number[]) {
  return Math.max(...numbers);
}

const numbers = [1, 2, 3];
  max([4, ...numbers]);
```

For object literals



```
const han = {
  firstName: 'Han'
};

const hanSolo = {
   ...han,
   lastName: 'Solo'
};
```

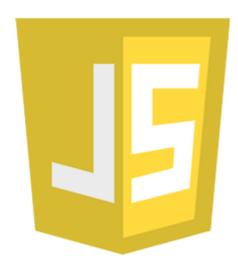
Object destructuring



```
function print({ name, age }: any) {
  console.log(`${name} is ${age}`);
}
print({ name: 'Obiwan', age: 68 });

function han() {
  return {
    name: 'Han Solo'
    };
}
const { name: hansName } = han();
hansName; // => 'Han Solo'
```

You can use object destructuring to mimic 'named parameters'.



Array destructuring

```
const list = [1, 2, 3];
const [a, , b] = list;
a; // => 1
b; // => 3
```

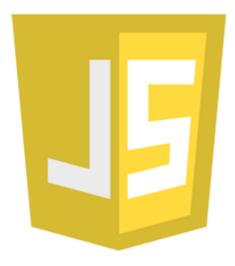


Destructuring examples

Destructuring also works "the other way".

```
const list = [3, 4, 5];
const fullList = [1, 2, ...list];
// => [1, 2, 3, 4, 5]

const defaults = { pretty: true, colors: true, logLevel: 'info' };
const overrides = { logLevel: 'debug' };
const options = { ...defaults, ...overrides };
// => { pretty: true, colors: true, logLevel: 'debug'}
```



Nullish coalescing

New in JavaScript: nullish coalescing

```
const personProperties = person.properties ?? {};
```

Using A ?? B returns A, if it's not *nullish*, otherwise B.

```
const firstName =
  person.properties !== null && person.properties !== undefined
  ? person.properties
  : {};
```

Similar to | but generally less error prone

```
const age = person.age || null;
// Can someone spot the bug here?
```



Optional chaining

New in JavaScript: optional chaining

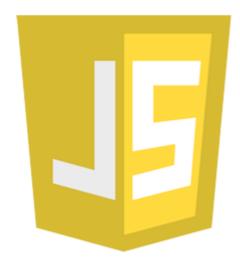
```
const firstName = person.properties?.firstName;
```

Use .? to evaluate an expression *optionally*, when the previous property in the *chain* is not *nullish*.

```
// Equivalent to
const firstName =
```

You can chain as many as you want.

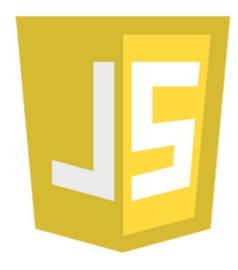
```
const foo = bar?.baz?.qux?.quux;
```



Optional chaining (2)

Optional chaining is also supported for functions, methods and property access.

```
log?.('Only log when it is available');
han.shootFirst?.();
arr?.[0];
```



Template literals

Normal string interpolation with

```
const name = 'foo';
const bar = 3;
const url = `https://example.com?q=${name}&s=${bar + 1}`;
```

But you can also create your own interpolation.

```
function get(stringParts: TemplateStringsArray, ...args: any[]) {
   // TODO: perform HTTP Get
}

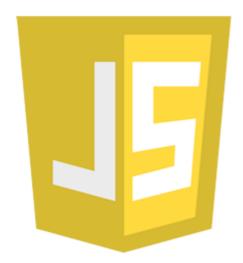
export const name = 'foo';
const bar = 3;
const response = get`https://example.com?q=${name}&s=${bar + 1}`;
```

Even more

http://es6-features.org/

Fat arrow functions

You can use () => to create a fat arrow function



```
const f = (x: number) => {
    return x + x;
}
```

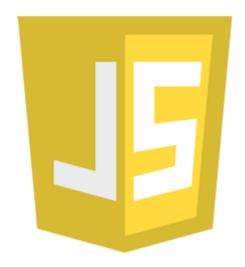
Omit return for one-liners

```
const f = (x: number) => x + x;
```

More on this later

Fat arrow functions

You can use () => to create a fat arrow function



```
const f = (x: number) => {
    return x + x;
}
```

Omit return for one-liners

```
const f = (x: number) => x + x;
```

More on this later