React.js

Introduction to ES6

Module Learning Objective

- Introduction to ES6
- The let and const
- The arrow functions
- Interface
- Classes
- Inheritance using extends
- Spread Operator
- Modules

ECMAScript

ECMAScript stands for **E**uropean **C**omputer **M**anufacturers **A**ssociation Script

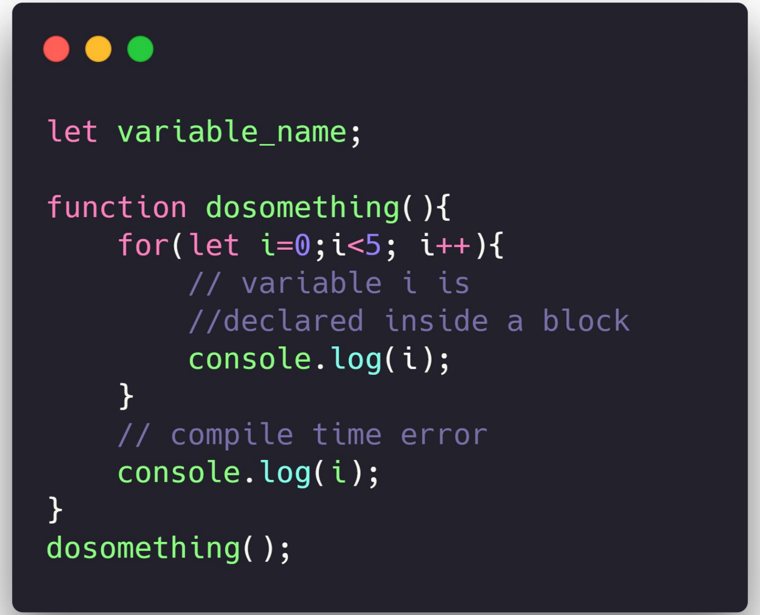
It lays out the specification and basic rules of language on which JavaScript is based on

- ES6 / ECMAScript 2015 was the second major release for JavaScript.
- Also known as JavaScript 6 as it is majorly derived from Netscape’s JavaScript
- ECMAScript has laid out a core object-oriented language to which Objects of any particular domain, or documents could be added

The let

- ES6 provides a new way of declaring a variable by using the let keyword. The **let** keyword is similar to the var keyword, except that these variables are **blocked-scope.**

```
let b = 20;

console.log(window.b); // undefined
```

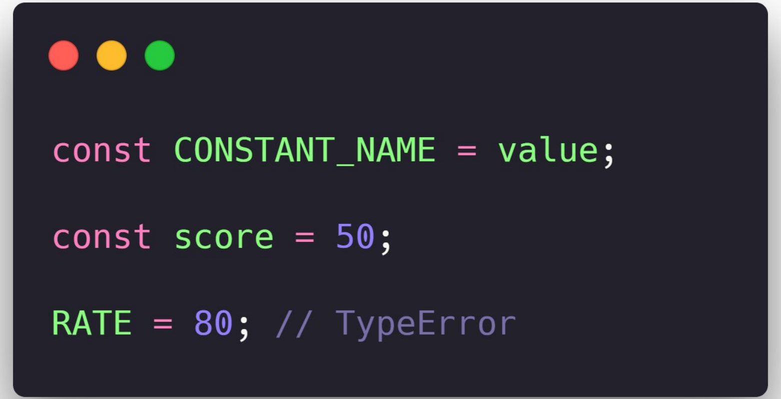


```
let variable_name;

function dosomething(){
    for(let i=0;i<5; i++){
        // variable i is
        //declared inside a block
        console.log(i);
    }
    // compile time error
    console.log(i);
}
dosomething();
```

The const

- ES6 provides a new way of declaring a constant by using the **const** keyword. The const keyword creates a **read-only** reference to a value.
- Like the let keyword, the const keyword declares blocked-scope variables. However, the block-scoped variables declared by the **const** keyword can't be **reassigned**.

const keyword are "**immutable**"

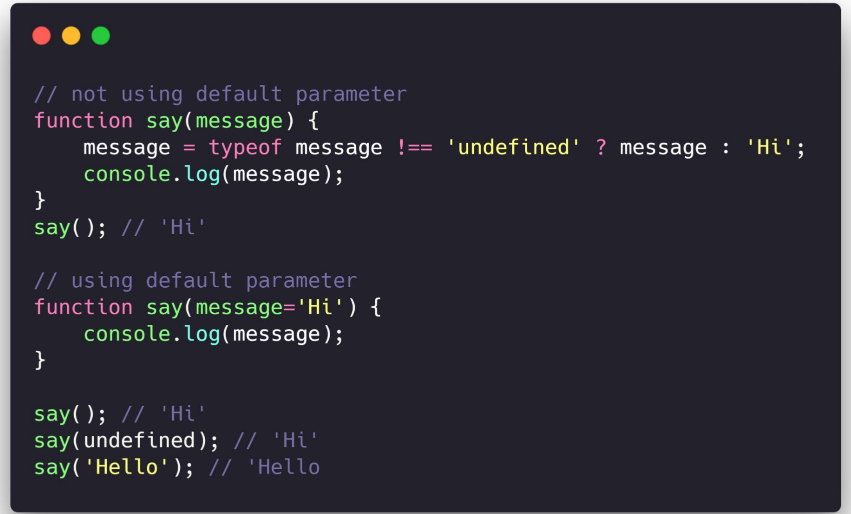


```
const CONSTANT_NAME = value;

const score = 50;

RATE = 80; // TypeError
```

Default Parameters

- In JavaScript, default function parameters allow you to initialize named parameters with default values if no values or undefined are passed into the function.

- **Argument** and **parameter** interchangeably. However, by definition, parameters are what you specify in the function declaration whereas the arguments are what you pass to the function.



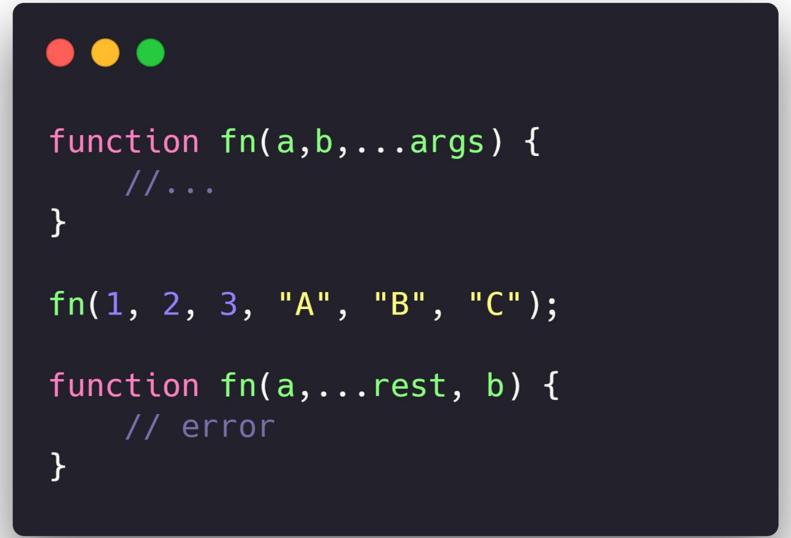
```
// not using default parameter
function say(message) {
    message = typeof message !== 'undefined' ? message : 'Hi';
    console.log(message);
}
say(); // 'Hi'

// using default parameter
function say(message='Hi') {
    console.log(message);
}

say(); // 'Hi'
say(undefined); // 'Hi'
say('Hello'); // 'Hello
```

Reset Parameters

- ES6 provides a new kind of parameter so-called rest parameter that has a prefix of three dots (...). A rest parameter allows you to represent an indefinite number of arguments as an array.

- The last parameter (args) is prefixed with the three-dots (...). It's called a rest parameter (...args).

Notice that the rest parameters must appear at the end of the argument list

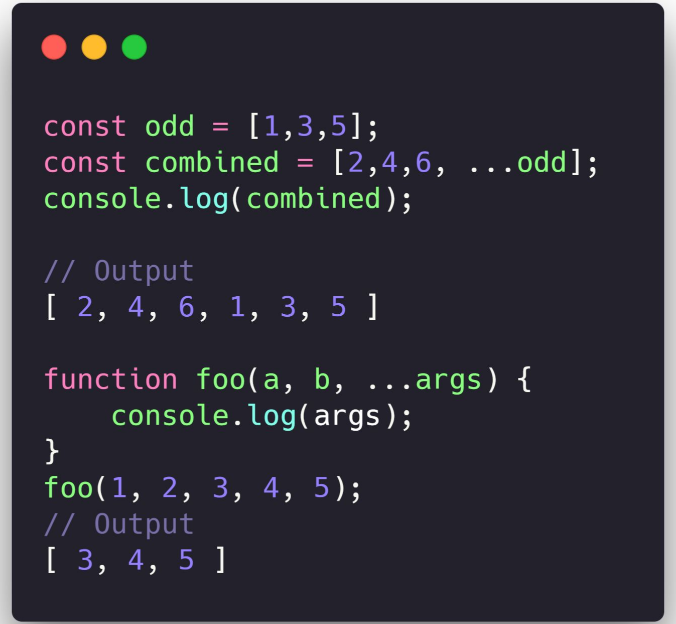


```
function fn(a,b,...args) {
    //...
}

fn(1, 2, 3, "A", "B", "C");

function fn(a,...rest, b) {
    // error
}
```

Spread Operator

- ES6 provides a new operator called spread operator that consists of **three dots (...).** The spread operator allows you to spread out elements of an iterable object such as an **array,a map,** or **a set**

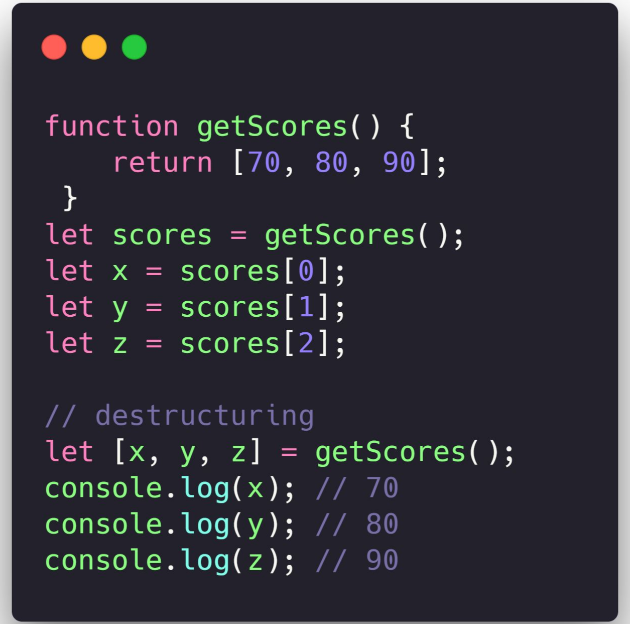
- ES6 also has the three dots (...) which is a rest parameter that collects all remaining arguments of a function into an array.



```
const odd = [1,3,5];
const combined = [2,4,6, ...odd];
console.log(combined);

// Output
[ 2, 4, 6, 1, 3, 5 ]

function foo(a, b, ...args) {
    console.log(args);
}
foo(1, 2, 3, 4, 5);
// Output
[ 3, 4, 5 ]
```

Destructuring

- ES6 provides a new feature called destructing assignment that allows you to destructure properties of an object or elements of an array into individual variables.

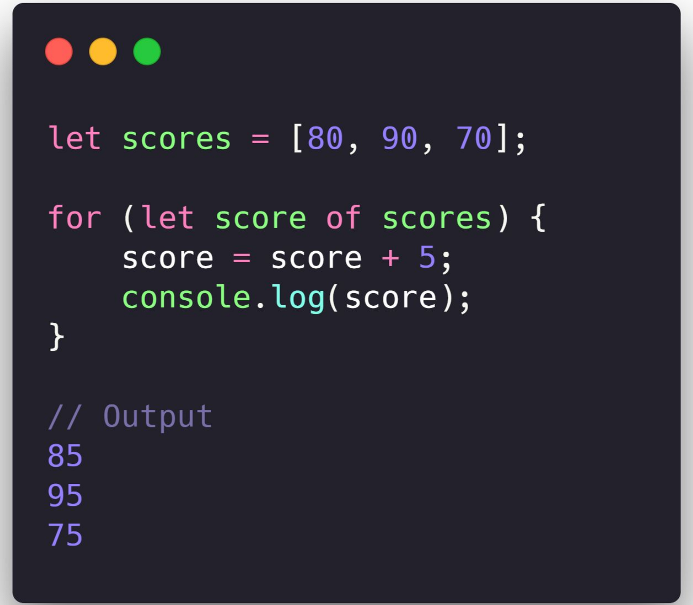


```
function getScores() {
    return [70, 80, 90];
 }
let scores = getScores();
let x = scores[0];
let y = scores[1];
let z = scores[2];

// destructuring
let [x, y, z] = getScores();
console.log(x); // 70
console.log(y); // 80
console.log(z); // 90
```

For...of loop

ES6 introduced a new construct **for...of** that creates a loop that iterates over iterable objects such as:

- Built-in Array, String, Map, Set, ...
- Array-like objects such as arguments or NodeList
- User-defined objects that implement the iterator protocol.



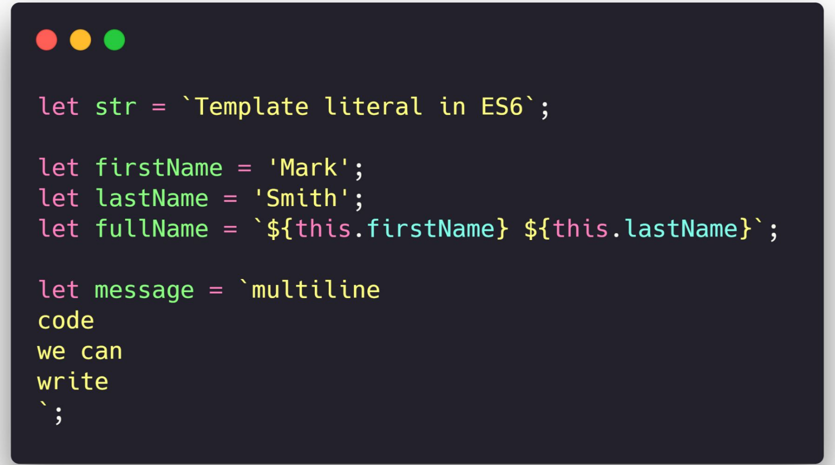
```
let scores = [80, 90, 70];

for (let score of scores) {
    score = score + 5;
    console.log(score);
}

// Output
85
95
75
```

Template literals

Prior to ES6, you use **single quotes (')** or **double quotes (")** to wrap a string literal. And the strings have very limited functionality.

In ES6, you create a template literal by wrapping your text in **backticks (`)** and you get the following features:

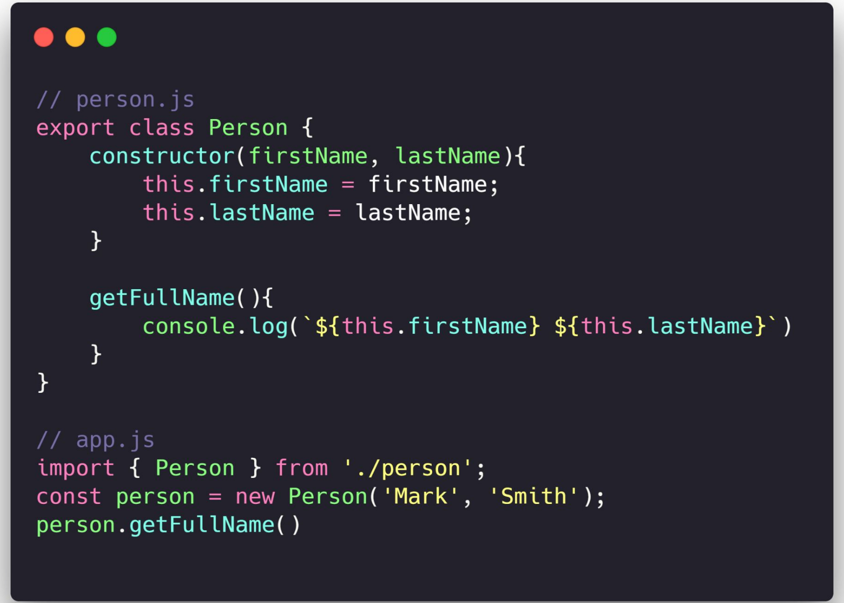
- A multiline string: a string that can span multiple lines.
- String formatting: the ability to substitute part of the string for the values of variables or expressions. This feature is also called string interpolation.
- HTML escaping: the ability to transform a string so that it is safe to include in HTML.



```
let str = `Template literal in ES6`;

let firstName = 'Mark';
let lastName = 'Smith';
let fullName = `${this.firstName} ${this.lastName}`;

let message = `multiline
code
we can
write
`;
```

Modules

- An ES6 module is a JavaScript file that executes in strict mode only. It means that any variables or functions declared in the module won't be added automatically to the global scope.

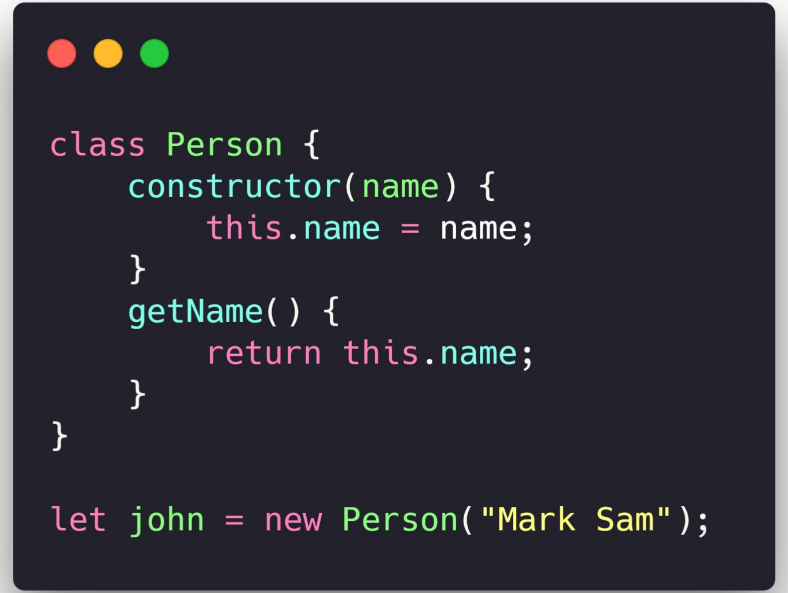


```javascript
// person.js
export class Person {
    constructor(firstName, lastName){
        this.firstName = firstName;
        this.lastName = lastName;
    }

    getFullName(){
        console.log(`${this.firstName} ${this.lastName}`)
    }
}

// app.js
import { Person } from './person';
const person = new Person('Mark', 'Smith');
person.getFullName()
```

Class

- A JavaScript class is a blueprint for creating objects. A class encapsulates data and functions that manipulate data.

- Unlike other programming languages such as Java and C#, JavaScript classes are syntactic sugar over the prototypal inheritance. In other words, ES6 classes are just special functions.

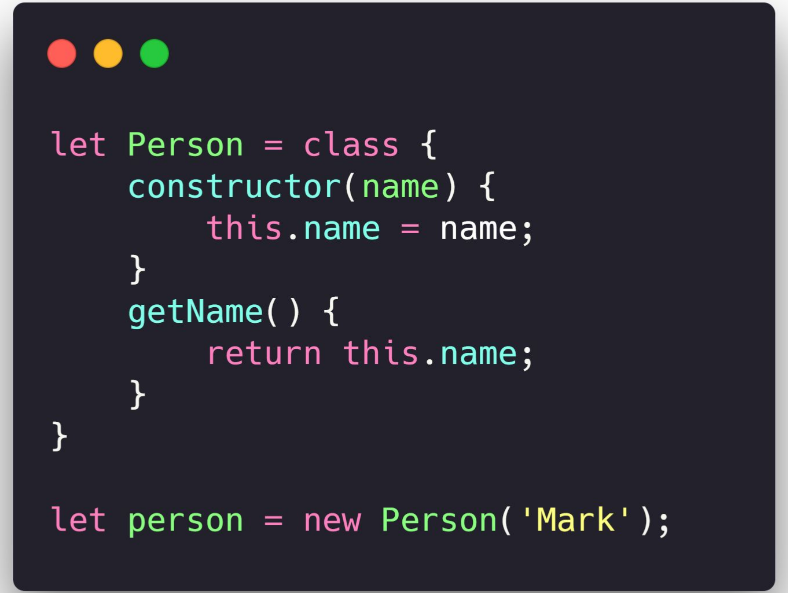


```
class Person {
    constructor(name) {
        this.name = name;
    }
    getName() {
        return this.name;
    }
}

let john = new Person("Mark Sam");
```

Class expression

- Similar to functions, classes have expression forms. A class expression provides you with an alternative way to define a new class.

- A class expression doesn't require an identifier after the class keyword. And you can use a class expression in a variable declaration and pass it into a function as an argument.



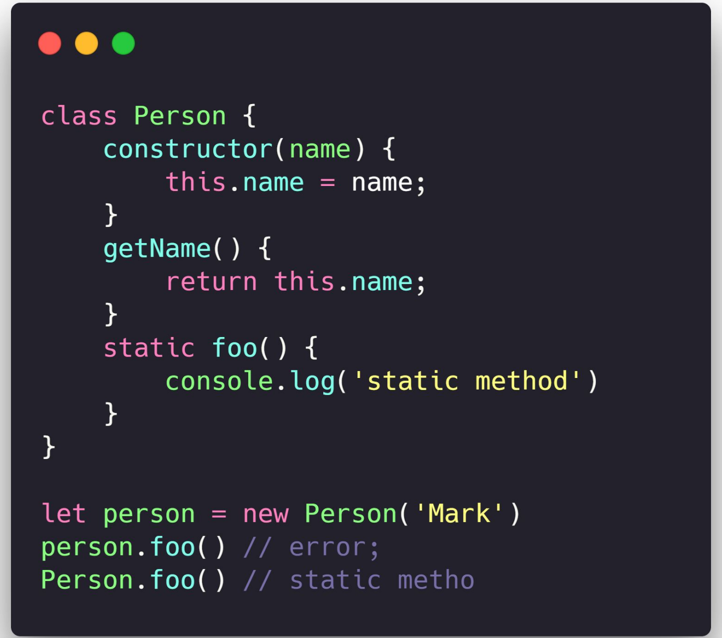
```
let Person = class {
    constructor(name) {
        this.name = name;
    }
    getName() {
        return this.name;
    }
}

let person = new Person('Mark');
```

Static methods

- By definition, static methods are associated with a class, not the instances of that class. Thus, static methods are useful for defining helper or utility methods.

- Prior to ES6, to define a static method, you add the method directly to the **constructor**.

- In **ES6**, you define static methods using the **static** keyword.

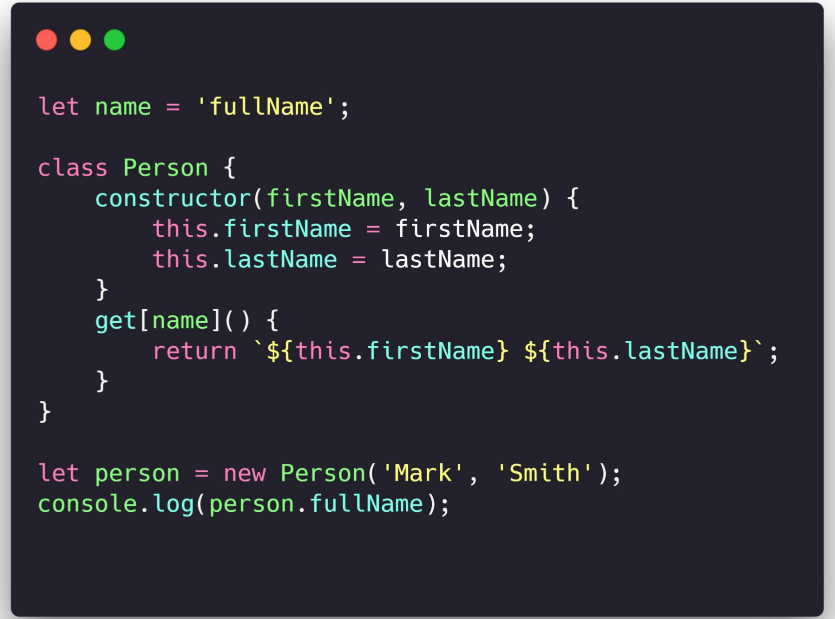


```
class Person {
    constructor(name) {
        this.name = name;
    }
    getName() {
        return this.name;
    }
    static foo() {
        console.log('static method')
    }
}

let person = new Person('Mark')
person.foo() // error;
Person.foo() // static metho
```

Computed Properties

- ES6 allows you to use an expression in brackets [ ]. It'll then use the result of the expression as the property name of an object.

- The get[name] is a computed property name of a getter of the Person class. At runtime, when you access the fullName property, the person object calls the getter and returns the full name.

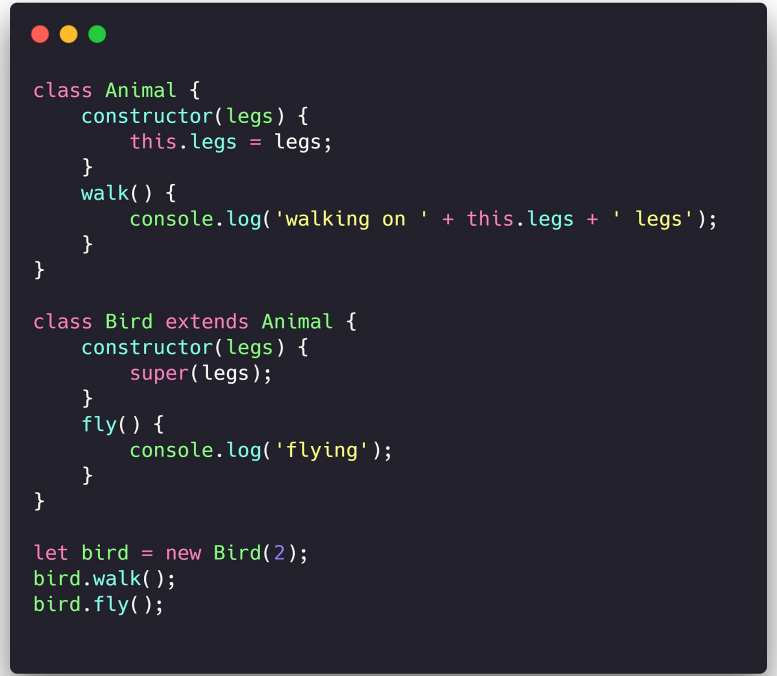


```
let name = 'fullName';

class Person {
    constructor(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    get[name]() {
        return `${this.firstName} ${this.lastName}`;
    }
}

let person = new Person('Mark', 'Smith');
console.log(person.fullName);
```

Inheritance using extends & super

- Prior to ES6, implementing a proper inheritance required multiple steps. One of the most commonly used strategies is the prototypal inheritance.

- ES6 simplified these steps by using the extends and super keywords.

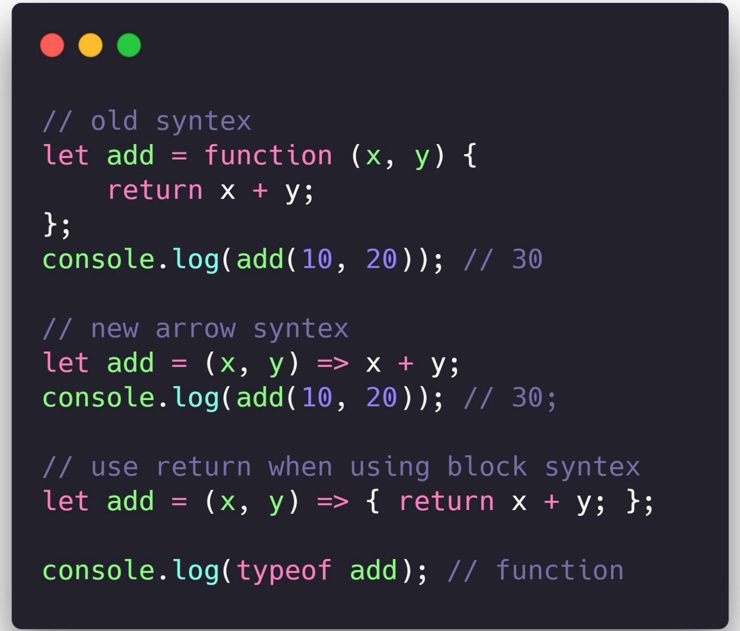


```
class Animal {
    constructor(legs) {
        this.legs = legs;
    }
    walk() {
        console.log('walking on ' + this.legs + ' legs');
    }
}

class Bird extends Animal {
    constructor(legs) {
        super(legs);
    }
    fly() {
        console.log('flying');
    }
}

let bird = new Bird(2);
bird.walk();
bird.fly();
```

Arrow function

ES6 **arrow** functions provide you with an alternative way to write a shorter syntax compared to the function **expression**.

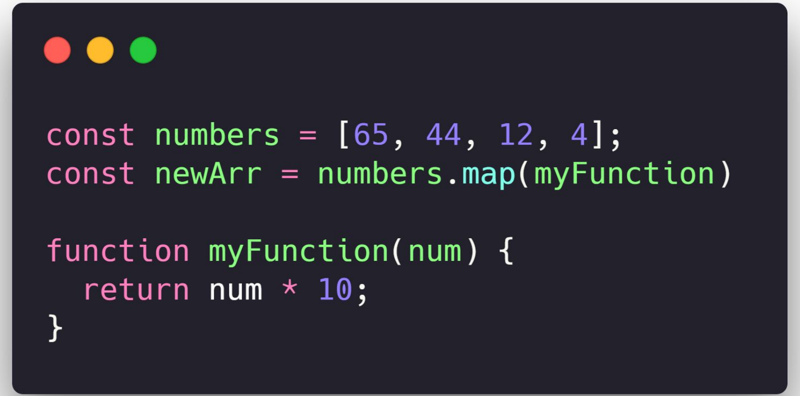
The following example is equivalent to the above **add()** function expression but use an **arrow** function instead



```
// old syntex
let add = function (x, y) {
    return x + y;
};
console.log(add(10, 20)); // 30

// new arrow syntex
let add = (x, y) => x + y;
console.log(add(10, 20)); // 30;

// use return when using block syntex
let add = (x, y) => { return x + y; };

console.log(typeof add); // function
```

Map function

- The map() method creates a new array with the results of calling a function for every array element.
- The map() method calls the provided function once for each element in an array, in order.
- map() does not execute the function for empty elements.
- map() does not change the original array.



```javascript
const numbers = [65, 44, 12, 4];
const newArr = numbers.map(myFunction)

function myFunction(num) {
  return num * 10;
}
```