

**ES6 Classes**:

So we learned all about objects and prototypical inheritance in JavaScript over the last few sections.

Now in ES6 or ES2015 (*modern version of JavaScript*), there is a new way to create objects and implement inheritance by using *classes*.

But these classes are not like the classes that we have in languages like C# and Java. *They are basically syntactic sugar over prototypical inheritance*.

So it is of utmost importance that we understand prototypical inheritance first before moving on to *ES6 classes which are cleaner and simpler*.

Here we have this constructor function that we have seen throughout the course, we have this radius property as well as the draw method.

function Circle(radius) {

  this.radius = radius;

  this.draw = function () {

    console.log("draw");

  };

}

Now we will see how can we write this code using ES6 classes.

We start with the name of the class prefixed with *class* keyword and then curly braces.

class Circle{

  🡪 *body of the class*

}

This is what we call the body of the class and *inside this body we can define properties and methods and one special method is a* ***constructor*** *which we use it to initialize objects*, just like the Circle constructor function we saw before.

In this method we pass, *radius* parameter and set the radius property on the new object instance that is created.

class Circle {

  constructor(radius) {

    this.radius = radius;

  }

}

If we want *to define a method, we can do it inside body of the class instead of constructor*.

class Circle {

  constructor(radius) {

    this.radius = radius;

  }

  draw() {

    console.log("draw");

  }

}

Note: To define a method inside an ES6 class, we don’t need to use function keyword, just add the name of method with parentheses and curly braces.

Now we can create circle objects just like before.

class Circle {

  constructor(radius) {

    this.radius = radius;

  }

  draw() {

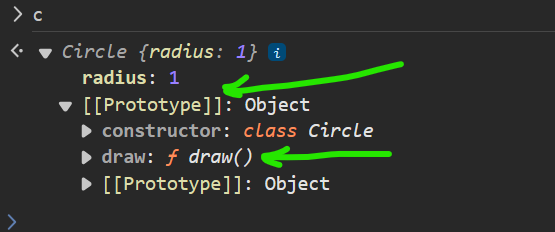
    console.log("draw");

  }

}

const c = new Circle(1); 🡪 *circle object*

In the console we can see the circle object with the radius property and the draw method in its prototype. So *all the methods we define in the body of the class will end up in the prototype of the circle object*.



Note: if *we don’t want a method to end on the prototype of an object, we need to define it in the constructor* just like before.

class Circle {

  constructor(radius) {

    this.radius = radius;

    this.move = function () {}; 🡪 *on the object instance*

  }

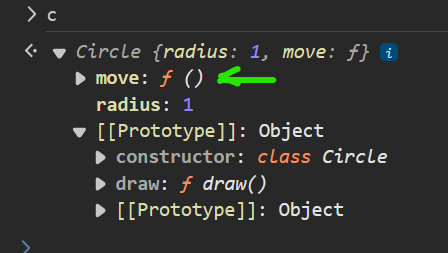
  draw() {

    console.log("draw"); 🡪 *on the prototype*

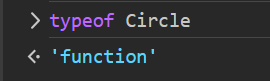
  }

}

So here we have *move* method on the object instance and draw method on the prototype.



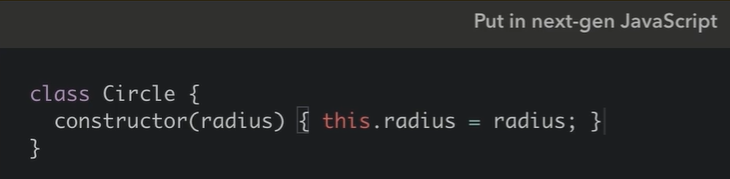
Let’s see type of Circle class, for fun!

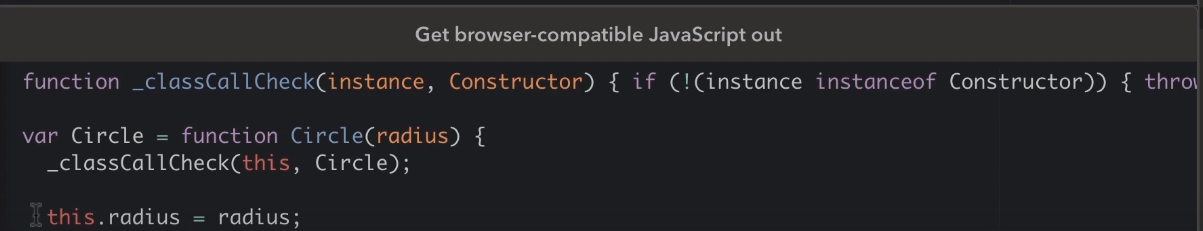


That’s a function! that’s why in the beginning we said classes are essentially constructor functions.

Still don’t believe it…? Let’s head over to <https://babeljs.io>

*Babel is a JavaScript compiler to which we give our modern JavaScript code and it compiles it down to ES5 code that all browsers understand*.

 🡪🡪

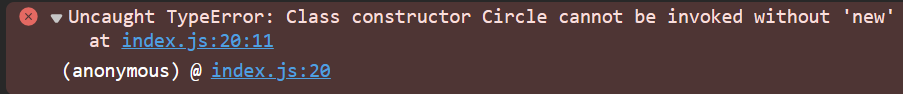
****

Note: This above snapshot is taken from lecture, as on 2024, we are getting same class based syntax in both blocks.

Continued…

The function \_*classCallCheck*(instance, Constructor) *job is to make sure that we have used new operator*. If we don’t its going to yell at us.

const c = Circle(1); //*remove new from here*



So classes in ES6 enforce the use of new operator.

**Hoisting**:

In JavaScript there are two ways two define functions. We can use the function declaration syntax or function expression syntax.

//function declaration

function sayHello() {

}

//Function expression

const sayGoodbye = function(){

};

Note: by convention we should end a function expression with a semicolon, because in JavaScript all expressions end with ;

There is one critical difference between function declaration and a function expression in JavaScript.

*Function declarations are hoisted which means they are raised to the top of the code, so I can call such a function before I declare it*.

sayHello();

//function declaration

function sayHello() {

}

-------------------->

//*it actually becomes like this*

//function declaration

function sayHello() { //*it moves to the top or say hoisted!*

}

sayHello();

In contrast, function expressions are not hoisted, so if we call *sayGoodbye* function before, we get an error.

sayGoodbye();

//Function expression

const sayGoodbye = function () {

};



Therefore function expressions are not hoisted (*it’s true for all types of expression like const number =1; which is a primitive type expression*).

When it comes to classes, we can define them using a class declaration or class expression syntax.

//class Declaration

class Circle {

}

//class Expression

const Square = class {

};

Unlike functions, class declaration and class expressions are not hoisted