# DAY 3 – ES6 (ECMA Script 2015)

## Let Const and Block scoping

No variable hoisting, no more depend on function scope, scope defined within curly braces.

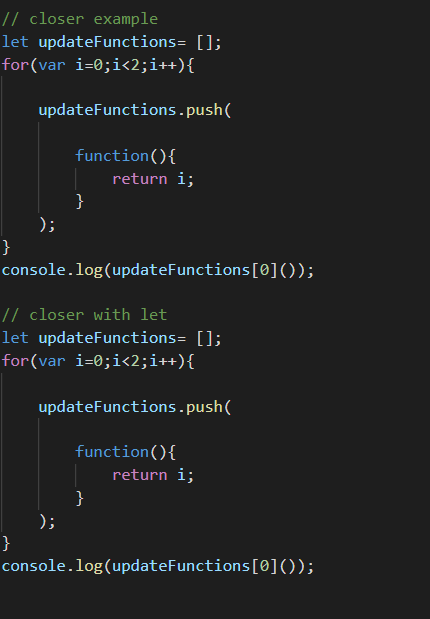
Example of Let

Console.log(product);// undefined

Var product = 10;

Console.log(product);// error

Let product = 20;

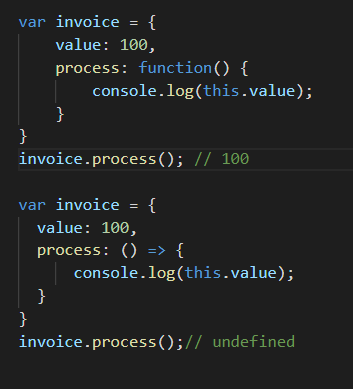


## Arrow Function

An **arrow function expression** is a syntactically compact alternative to a regular [function expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function), although without its own bindings to the [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this), [arguments](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments), [super](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super), or [new.target](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new.target) keywords. Arrow function expressions are ill suited as methods, and they cannot be used as constructors.

Main benefit of Arrow function is context of this. With call and bind you can’t change the context of this.

For Arrow function it gets this of executing context.



## Default Function Parameters

Var getProduct = function(a=10,b=10){

Return a+b;

}

getProduct();

Default parameter can reference to outside variable and previously defined variable, and if you pass undefined and there is default value mentioned in function definition then default will be used.

## Rest and Spred…

The **rest parameter** syntax allows us to represent an indefinite number of arguments as an array.Object Literal Extension.

function f(*a*, *b*, ...*theArgs*) {

// ...

}

**Spread syntax** allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected.

function myFunction(x, y, z) { }

const args = [0, 1, 2];

myFunction(...args);

Spread as array

const parts = ['shoulders', 'knees'];

const lyrics = ['head', ...parts, 'and', 'toes'];

// ["head", "shoulders", "knees", "and", "toes"]

## Object literal Extension

let a = 'foo',

b = 42,

c = {};

// Shorthand property names (ES2015)

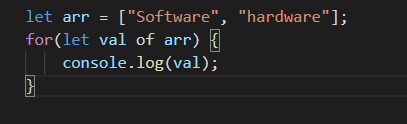
let o = {a, b, c}

// In other words,

console.log((o.a === {a}.a)) // true

## For.. of Loop

New loop to iterate and read element



## Octal and Binary literals

## 

## Template Literals

Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them.

`string text`

`string text line 1

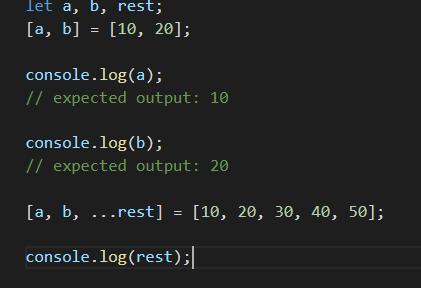
string text line 2`

`string text ${*expression*} string text`

*tag*`string text ${*expression*} string text`

## Destructuring

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.



## Module and Classes

### Module Basic

**Installation**

Use of native JavaScript modules is dependent on the [import](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import) and [export](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export) statements, the browser compatibility

Module supports only on server( as cors denied accces to local files) so create a local server

Go to the project directory run below command in CMD/npm cli

Step 1- $npm init;

Step2- $npm install http-server -g; // install local server locally

Step 3- $http-start // to Run the server

Change in index.html from type= “text/javascript” => type=”module”

Create a directory for Modules

IF type=”module” is not supported by your browsers then you have to use some module loader plugin to work with ex- <https://blog.revillweb.com/using-es2015-es6-modules-with-babel-6-3ffc0870095b>

### Classes

Classes are not hoisted as normal variable or function.

JavaScript classes, introduced in ECMAScript 2015, are primarily syntactical sugar over JavaScript's existing prototype-based inheritance. The class syntax does not introduce a new object-oriented inheritance model to JavaScript.

Defining classes

Classes are in fact "special functions", and just as you can define function expressions and function declarations, the class syntax has two components: class expressions and class declarations.

One way to define a class is using a **class declaration**. To declare a class, you use the class keyword with the name of the class ("Rectangle" here).

class Rectangle {

constructor(height, width) {

this.height = height;

this.width = width;

}

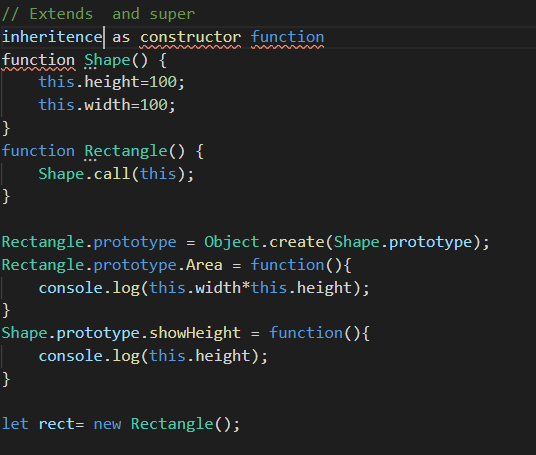
}

#### Hoisting

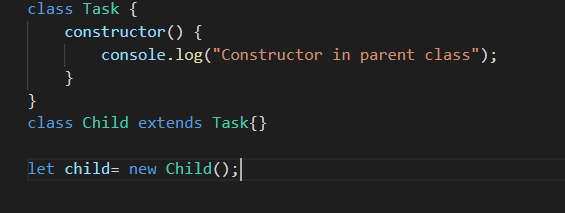
An important difference between **function declarations** and **class declarations** is that function declarations are [hoisted](https://developer.mozilla.org/en-US/docs/Glossary/Hoisting) and class declarations are not. You first need to declare your class and then access it, otherwise code will throw a [ReferenceError](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ReferenceError):

### Extends and Super

Inheritance in Constructor pattern (Prototypical inheritance)

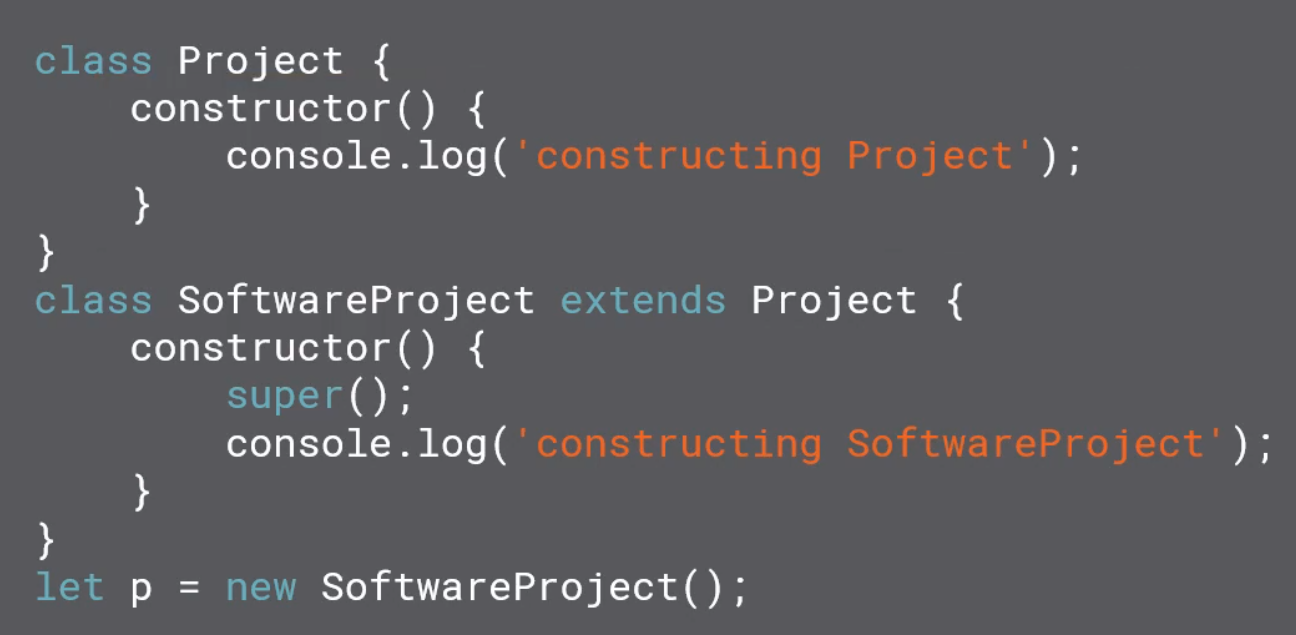


Inheritance in ES6 is with the keywords Extends .

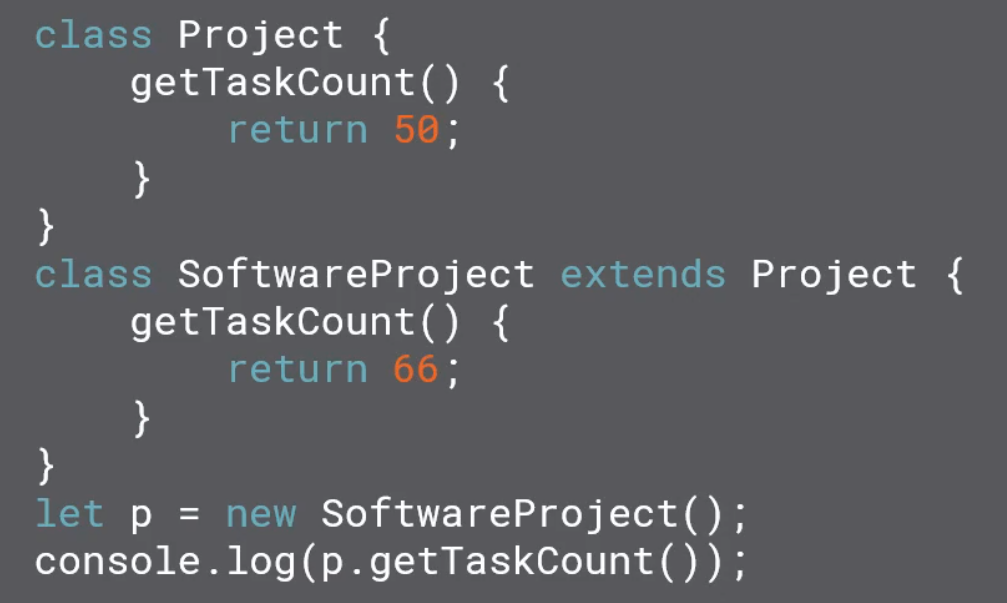


Super keyword

It works similar to other languages (Java), If you define a constructor in child class you have to call super first otherwise it will throw error on instantiation (thi)



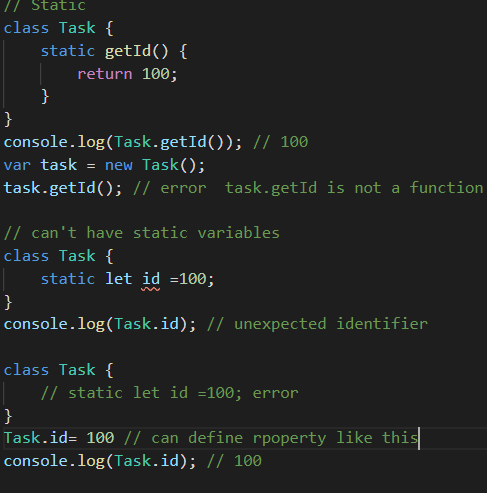
Overriding



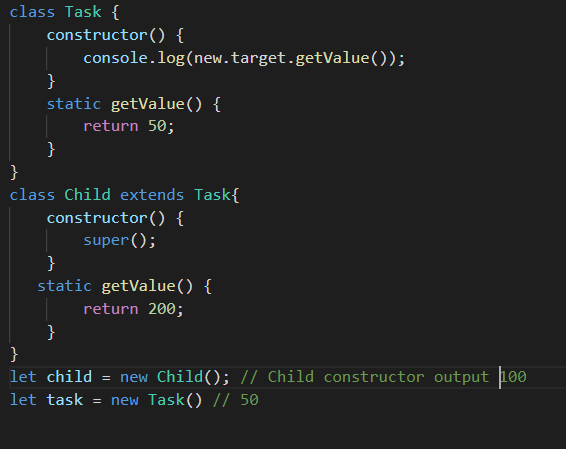
### Static Member

We can have Static method in es6 but can’t define static variables

Static method attached to class and no instance have access to it.

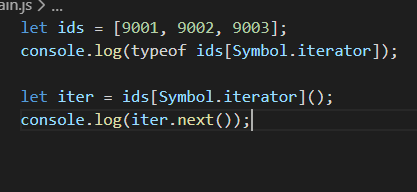


### New.target

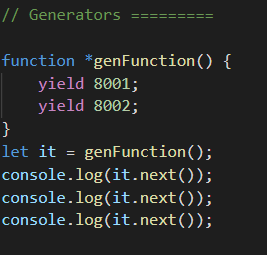


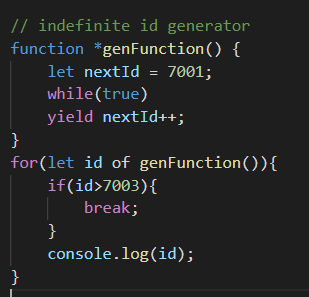
## Iterators and Generators

Iterators ->



Generators -> function \* genFunction

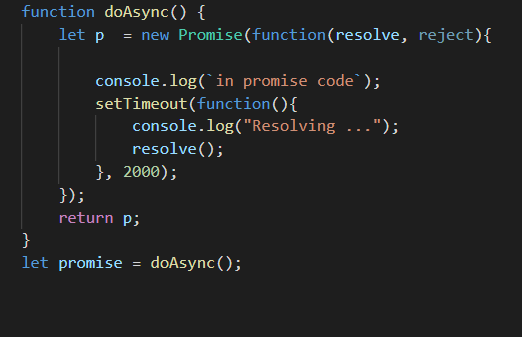


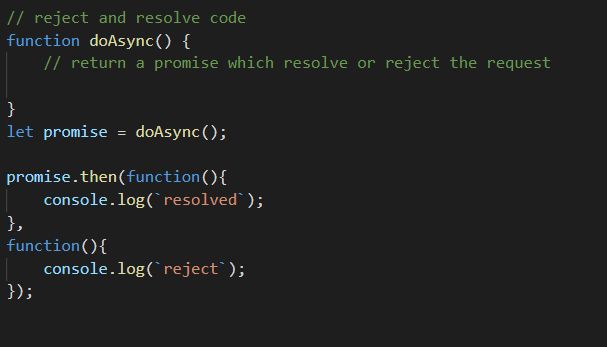


## Promise

It gives ability to do

asynchronous programming natively.





Promise chaining can also be done.