**Exercise03\_01\_01 – Step 1**

In this Exercise, we will transition to ES6 in Node.js.



1. Create a folder named ***Exercise03\_01\_01***. Retrieve the ***Exercise03\_01\_01.zip*** file from Google Classroom. Extract the files to your project folder. Open your IDE to the project folder. From a terminal run the following command to install the dependencies from package.json:  
   ***npm install***
2. Open the ***package.json*** file. You can see a list of dependencies. Take note that ***express*** is not one of them. We will take a look at a node ***server*** file which is not using express shortly. In the meantime, you will notice that there is a ***start*** script. The start script will have the command to execute when an ***npm start*** is issued at the terminal, or from your IDE if possible. In this project, it just says to run node, with a file of ***.*** as the argument. That tells it to use the entry point specified in the ***main*** name value pair, which is ***src/server.js***.
3. Go into the ***/src*** folder and open the ***server.js*** file. let’s take a look around at a raw node.js file that builds a server without using ***express***. Interesting. Just a quick look, this is not the point of this unit. We need to learn about ***JavaScript ES6***.
4. From a terminal, or the IDE if possible, let’s run an ***npm start***. We should be listening on port ***8080***. We can open up a browser and take a look at the starting point of this application.
5. The demo project is called ***Pizza*** ***Market***. It is a ***stock*** ***market*** ***simulation*** that trades in ***pizza*** slices. This web application is built entirely in node, with the back end calculating the new quotes.  
     
   You can ***open*** and ***close*** the market, which begins generating some random daily stock numbers for different types of pizza slices. This chart is the ***aggregate*** of all the pizza slices daily. You can click on individual pizza slices and see the data. Click back on the Pizza Market logo to get back to the aggregate display.  
     
   All of these quotes are generated on the back end by our node application. Take a look at the ***node*** ***console*** and you can watch the quotes updating when the market is open.  
     
   You can click close market to stop the simulation and open market to start it again. Pizza Market is now all set up, so you're ready to move on to learning ***ES6***.

**Exercise03\_01\_01 – Step 2**



1. Go to the ***/src*** folder and let’s begin our conversion to ES6 with ***server.js***. This file is the main entry point to the application. We will look for any ES5 function syntax to convert to ES6 ***arrow functions***. Go to the ***server.register()*** function, where we are passing in a ***callback***. We can convert to ES6 by removing the ***function*** keyword and adding a ***fat*** ***arrow*** after the parameter list. We can remove the ***parentheses*** around a single parameter list. It is sometimes considered bad practice but very readable here.  
   server.register([require('inert'), require('vision')], ***err =>*** {  
    if (err) throw err;  
   Much cleaner and more readable. Run the server with ***npm*** ***start*** and the browser to check syntax and logic.
2. Let’s do the same conversion with the ***server.start()*** function as follows:  
    server.start(***err =>*** {  
    if (err) throw err;  
   Again, much cleaner and more readable. Run the server with ***npm*** ***start*** and the browser to check syntax and logic.
3. Let’s try a slightly more complex function. Open the ***/src/lib*** subfolder and go to the ***fluxGen.js*** file. The module consists of one function and a ***module.exports*** assignment. Notice the interesting way in which the export is done. First we will convert it to an ***anonymous*** ***arrow*** ***function*** by removing the ***function*** keyword and function ***name***:  
   ***(seed, times, variability, positivity) => {***
4. Because the function no longer has a name, the ***module.exports*** assignment is now broken. Let’s directly export the arrow function, remembering to put a ***semicolon*** after the last curly brace because it is now an assignment statement:  
   ***module.exports =*** (seed, times, variability, positivity) => {  
    var output = [],  
    current = seed,  
    change;  
     
    for (var i = 0; i < times; i++) {  
    change = (Math.random() \* variability).toFixed(0);  
    if ((Math.random() \* positivity) <= (positivity / 2)) {  
    change = -change;  
    } else {  
    change = +change;  
    }  
    current += change;  
    output.push(current);  
    }  
    return output;  
   }***;***  
   Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.
5. This is still working as an ***anonymous*** arrow function because the modules, like ***/src/models/pizza.js***, that use the function, do the following ***require()*** so that the function can still be called by name:  
   ***var fluxGen = require(’../lib/fluxGen’);***

**Exercise03\_01\_01 – Step 3**



1. We will examine the differences in ***scoping*** , ***closure***, and ***hoisting*** in ES5 versus ES6. Open the ***/src/models*** subfolder and go to the ***pizza.js*** file. There is a constructor function named ***Pizza()*** which defines our pizza objects. There are two ***private*** ***methods*** in the constructor, ***addQuote()*** and ***getQuote()***. Private methods are inner functions of the constructor, inaccessible to the outside because of lexical scoping. They are regular functions, so ***lexically*** scoped. Notice that to get to the ***this*** reference, we have to store it in a lexically scoped ***variable*** declaration, at the top of the lexical scope, because of its ***closure*** characteristics. It closes around the ***callers***, addQuote() and getQuote(). We can remove it.:  
    ***~~var self = this;~~***
2. By changing to arrow functions, it is not necessary to use the extra variable declaration as the private functions will close around the ***parent***. Let’s start by converting **addQuote()** to an ***arrow*** ***function***, which we will also assign to a ***var*** for calling purposes. Due to the simplicity of the function, we can also remove ***parentheses*** and ***curly*** braces. We can also get rid of ***self***, and directly use the ***this*** reference:  
    ***var addQuote = quote => this.quotes.push(quote);***
3. Let’s repeat the same steps with ***getQuote()*** and we can also remove the ***return*** statement:  
    ***var getQuote = quoteIndex => this.quotes[quoteIndex];***
4. This has given us a ***hoisting*** problem, the ***var*** declarations of the functions are hoisted. But the assignments of their definitions are left behind. So they are ***undefined***. We must move the declarations up before the functions are actually called:   
    this.quotes = quotes || [this.startingQuote];  
    ***// private methods  
    var addQuote = quote => this.quotes.push(quote);  
    var getQuote = quoteIndex => this.quotes[quoteIndex];***Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.

**Exercise03\_01\_01 – Step 4**



1. Open the ***/src/models*** subfolder and go to the ***pizza.js*** file. In the constructor function ***Pizza()***, the two last parameters are treated as ***optional***, ES5 style. This is done in the property initialization list by using ***logical*** OR ( || ) constructs:  
    this.variability = variability || getRand();  
    this.positivity = positivity || getRand();
2. In ES6, we can do a cleaner and more readable implementation by using ***default*** ***parameters*** as follows:  
   function Pizza (startingDate, quotes, ticker, name, startingQuote,   
    ***variability=getRand()***, ***positivity=getRand()***) {
3. We can now modify the property assignment statements to remove the logical OR expressions:  
    ***this.variability = variability;  
    this.positivity = positivity;***Much cleaner and more readable. Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.

**Exercise03\_01\_01 – Step 5**



1. Open the ***/src/models*** subfolder and go to the ***pizza.js*** file. In the constructor function named ***Pizza()***,let’s modify the function ***signature*** to indicate that the first two parameters are ***required***, and the remaining ones will be ***rest*** parameter as follows:  
   ***function Pizza (startingDate, quotes, ...pizzaProps) {***
2. The ***ticker***, ***name***, ***startingQuote***, ***variability***, and ***positivity*** variables are going to be called in the order that they are assigned, if used. So we can use regular ***array*** syntax for them. Of course, if we call them in the wrong order, this is going to be a disaster, which is why one should be very careful in the use of rest parameters. Modify as follows:  
    ***this.ticker = pizzaProps[0];  
    this.name = pizzaProps[1];  
    this.startingQuote = pizzaProps[2];  
    this.variability = pizzaProps[3] || getRand();  
    this.positivity = pizzaProps[4] || getRand();***Quite an elegant solution. We should not even have to look for the place(s) where the constructor may be called, anything already there should work based on the original code. Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.

**Exercise03\_01\_01 – Step 6**



1. Open the ***/src/lib*** subfolder and go to the ***dataStore.js*** file. Find the ***initPizzas()*** function. The function does a ***require()*** to get an array of arrays of pizza arguments. It uses them to construct new pizza objects, and it is very repetitive:  
    ***realPizzas[pizza[0]] = new Pizza(startingDate,   
    data.quotes[pizza[0]], pizza[0], pizza[1], pizza[2], pizza[3],   
    pizza[4]);***
2. We can clean up the constructor call really nicely with a ***spread*** operator. It will convert the pizza array elements into a sequence of individual elements for us to pass into the ***Pizza*** constructor:  
    ***realPizzas[pizza[0]] = new Pizza(startingDate,   
    data.quotes[pizza[0]], ...pizza);***Really cleans up that code. Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.

**Exercise03\_01\_01 – Step 7**



1. Open the ***/src/models*** subfolder and go to the ***pizza.js*** file. Find the ***Pizza()*** function, which acts as an ES5 ***constructor***. To convert this to an ES6 class, substitute ***class*** for the ***function*** keyword. Then place an opening curly brace after the identifier ***Pizza*** and add a line feed. Use the ***constructor*** keyword before the argument list. Finally, close off all of the ***this*** assignments with a closing curly brace:  
   ***class Pizza {  
    constructor(startingDate, quotes, ...pizzaProps) {  
    this.startingDate = startingDate;  
    this.ticker = pizzaProps[0];  
    this.name = pizzaProps[1];  
    this.startingQuote = pizzaProps[2];  
    this.variability = pizzaProps[3] || getRand();  
    this.positivity = pizzaProps[4] || getRand();  
    this.quotes = quotes || [this.startingQuote];  
    }***
2. Now let’s convert our methods to ES6. There is no way in ES6 to make a method ***private***, so we will add a comment that they are ***public***. For the ***getNext()*** function, we remove the ***this*** and ***function*** keywords, as well as the ***assignment*** operator. We must remove the ***semicolon***, because it is no longer an assignment statement:  
    ***// public methods  
    getNext () {*** var newQuote = fluxGen(this.getLast(), 1, this.variability,   
    this.positivity)[0];  
    addQuote(newQuote);  
    return newQuote;  
    ***}***
3. Let’s now convert the ***getLast()*** and ***getDatedQuotes()*** functions in the same way:  
    ***getLast ()*** {  
    return getQuote(this.quotes.length - 1);  
    ***}***  
     
    ***getDatedQuotes ()*** {  
    var quotesMap = {},  
    curDate = startingDate;|  
     
    this.quotes.forEach(function (quote) {  
    quotesMap[curDate] = quote;  
    curDate.setDate(curDate.getDate() + 1);  
    });  
     
    return quotesMap;  
    ***}***
4. Now let’s scroll up and take care of our ***private*** methods. We cannot use ***arrow*** function syntax inside a class, so we will need to convert them to regular functions. There is no real way to make them private, as in other languages, but Best Practice is to add an ***underscore*** at the beginning of their identifiers to signal other programmers that they are meant for ***private*** ***usage***. Don’t forget to put back the ***return*** keyword:  
    // private methods  
    ***\_addQuote (quote) {***  
    ***return*** this.quotes.push(quote)  
    }  
     
    ***\_getQuote (quoteIndex) {***  
    ***return*** this.quotes[quoteIndex];  
    }
5. Now we need to clean up the calls to these functions so they have the new ***underscore***:  
    getNext () {  
    var newQuote = fluxGen(this.getLast(), 1, this.variability,   
    this.positivity)[0];  
    ***this.\_addQuote***(newQuote);  
    return newQuote;  
    }  
     
    getLast () {  
    return ***this.\_getQuote***(this.quotes.length - 1);  
    }
6. We need to fix up a scope problem with the variable ***startingDate***:  
    getDatedQuotes () {  
    var quotesMap = {},  
    curDate = ***this.startingDate***;  
   That completes the conversion to ES6 ***class*** syntax. Run the server with ***npm*** ***start*** and the browser to check syntax and logic. Open the market, highlight some different pizzas, and close the market to make sure nothing is broken.