Lesson 4.2 - Arrow Functions Everywhere

So far in this lesson we have learned a bit about how arrow functions work. Let's write a little bit of code so that we can see how this might help us in the real world.

We're going to create a receipt calculator function that leverages this feature.

Let's go ahead and move into the directory that we established in the first lesson where we're going to hold all of our code for this tutorial and create a file called `calculator.js`. Once this is created go ahead and open it in your favorite text editor.

Let's start by defining some sample data that we can export for easy testing in the console:

```
export const items = [
  { price: 10, isTaxable: true },
  { price: 22, isTaxable: false },
  { price: 13.45, isTaxable: true }
];
```

Great! We've set up a simple data structure here that holds a list of items that somebody is purchasing. Each of these items includes the cost of the item (price) and whether or not the item is taxable.

Next we'll go ahead and start creating our function. Let's start with the shell of the function:

```
export function calculate() {

}
```

Now we've got a basic function, but unfortunately it doesn't do anything. Let's start by adding some parameters. There's a few things this function will need to know in order to be useful. The first thing it's going to need to know is the rate at which to apply tax to a particular item.

```
export function calculate(taxRate) {

}
```

Ok great, but we're going to need more. We also need to have a list of items to iterate through in order to calculate the total. Let's add that as the items variable:

```
export function calculate(taxRate, items) {

}
```

Awesome! Now we've got a function that takes two arguments! Let's write the meat of the function now:

```
export function calculate(taxRate, items) {
  return items.reduce((prev, curr) => {
    if (curr.isTaxable) {
      prev.total += curr.price;
      prev.totalTax += curr.price + (curr.price * taxRate / 100);
    } else {
      prev.total += curr.price;
    }
    return prev;
  }, {
    total: 0,
    totalTax: 0
  });
}
```

If you're not familiar with the `reduce` function that we used in the above code, feel free to check out how it's used here: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce

That looks like it will work, but I think we should leverage the `math` module that we created in the last lesson to make this easier! Let's import it and put it to work:

```
import { calculateTax } from './math';
```

Ok it's imported, now let's use it:

```
export function calculate(taxRate, items) {
  return items.reduce((prev, curr) => {
    prev.total += curr.price;
    prev.totalTax += calculateTax(curr.price, taxRate, curr.isTaxable);
    return prev;
  }, {
    total: 0,
    totalTax: 0
  });
}
```

Great! We've now got a function that we can use to calculate a total from a list of items. Let's open up the REPL and try it out!

```
$ babel-node
> var { items, calculate } = require('./calculate')
> calculate(8, items)
-> { total: 45.45, totalTax: 1.876 }
```

Cool! It works! We've now got a handy little function that will calculate our sub-total and the total tax for a list of items. Now we're getting somewhere!