ES6 Exercises

Jump to bottom

Varoon Sahgal - vn0silf edited this page on Sep 19, 2017 · 9 revisions

ES6 Exercises Instructions - Please Read:

Here are the ES6 specific exercises that we expect you to complete as part of your pre-work for the React and Redux Fundamentals Course. These exercises should take about 1 to 1.5 hours. That time does not include the time we expect you to put in reading both the ES6 and React Study Guides.

We also link to the appropriate sections in the ES6 study guide to help you complete the exercises below.

Please use JSFiddle to submit each exercise - we recommend creating an account if you don't already have one. You should also submit your answer to each exercise as a separate JSFiddle url. **So, you should have a total of 8 JSFiddle url's as there are 8 exercises.**

A quick note on debugging within JSFiddle: To see output from console.log() while still inside JSFiddle, go to External Resources on the left-side panel and add the following link for Firebug: https://getfirebug.com/firebug-lite-debug.js

Mandatory ES6 Exercises

1. Arrow Functions Exercise
2. Destructuring Exercise 1a
3. Destructuring Exercise 1b
4. ES6 Classes Exercise 1a
5. ES6 Classes Exercise 1b
6. Rest Operator Exercise
7. Spread Operator Exercise
8. Rest Operator Exercise 2

Arrow Functions Exercise

- Below we have a factorial function that clearly uses the 'function' keyword. **Your challenge exercise is to refactor the factorial function below to use ES6 fat arrow syntax.** Keep in mind that there isn't anything wrong with using the `function` keyword, but it does look better with the fat arrow syntax instead.

Fat arrow function rules

- Keep in mind the rules for fat arrow functions:

```
- For functions with just one argument, the parentheses around the argument
list are not required.
- For functions with just a single expression in the body, we can remove the
curly
  braces and 'return' keyword.
```

Factorial with "function" keyword:

```
const fact = function factorial(n) {
  if (n === 0) return 1;

  return n * factorial(n - 1);
}
```

Destructuring Exercise 1a

- Below we have some code that references `bio` *twice* inside the `isDeveloper` function.

```
const bio = {
  title: 'Developer',
  department: 'GEC'
};


function isDeveloper(bio) {
  var title = bio.title;
  var department = bio.department;
  return title === 'Developer' && department === 'GEC';
}
```

- **Your challenge exercise is to refactor the code used to reference the `title` and `department` properties. You should use *destructuring*.** Try to get the `isDeveloper` function down to a single line.

Destructuring and Map() Exercise 1b

- Suppose we have an array of arrays representing someone's shopping cart on Walmart.com. For example, our array of arrays would look something like this:

```
const cart = [
  [ 'Hersheys Bar', '1.00', '504' ],
  [ 'Almonds', '5.00', '321'],
  [ 'Lotion', '2.50', '287' ]
];
```

- Each array in our cart has the structure of `[item, price, sku]` - in that exact order. **Now, your challenge exercise is to convert this array of arrays into an array of *objects* instead.** Each object inside the array should have the keys `item`, `price`, and `sku` - along with the associated values for each key. The resulting array of objects should be assigned to a variable named `cartAsObjects`.

- Your data structure should have a setup like this (in this example we only have one object, but yours would have three):

```
const cartAsObjects = [{ item: 'Hersheys Bar', price: '1.00', sku: '504' }]
```

- So, to summarize, take the array of arrays (reproduced below) and convert it into an array of ***objects*** stored in an array called `cartAsObjects`. You must use both *array destructuring* and the *map* helper.

```
const cart = [
  [ 'Hersheys Bar', '1.00', '504' ],
  [ 'Almonds', '5.00', '321'],
  [ 'Lotion', '2.50', '287' ]
];
```

- Remember that to destructure arrays we must use the square brackets (the `[]`) instead of the curly braces (the `{}`).

ES6 Classes Exercise 1a

- Create an ES6 class called `Developer`. **Your challenge exercise is to do some basic initialization for instances of the Developer class inside the constructor.** Here is what you need to do specifically:

```
- The constructor will accept a 'profile' object that has 'name' and 'title'
properties.
  You need to assign those 'name' and 'title' properties to Developer as
well.
- Initialize the 'company' property of the Developer class to 'Walmart'.
- Create an instance of the class.
```

ES6 Classes Exercise 1b

- Now that we have our Developer class, your challenge exercise is to create a subclass of the `Developer` class called `JavascriptDeveloper`.
- The requirements for the `JavascriptDeveloper` class are:

```
- The JavascriptDeveloper class should have a `newTitle` method.  The only
argument to this
  method is another instance of the `JavascriptDeveloper` class.
- The instance of `JavascriptDeveloper` that is passed into `newTitle`
should have
  it's `title` changed to `Javascript Developer`.
- Create an instance of the class.
```

Rest Operator Exercise

- Refactor the function below to use the rest operator:

```
function product(v, w, x, y, z) {
  var numbers = [v,w,x,y,z];

  return numbers.reduce(function(acc, number) {
    return acc * number;
  }, 1)
}
```

Spread Operator Exercise

- Refactor the function below to use spread operator (arr1 and arr2 are arrays):

```
function join(arr1, arr2) {
  return arr1.concat(arr2);
}
```

Rest Operator Exercise #2

- Refactor the function below to use only the rest operator:

```
function unshift(array, x, y, z) {
  return [x, y, z].concat(array);
}
```

https://gecgithub01.walmart.com/developer-portal/react-redux-fundamentals.wiki.git