**Question 1**  
  
See question, I note to myself that this means revising my generally minimal javascript where knowledge has dropped to zero over time.

Decide not to set up a github project, cause files should be zipped, will regret this when ssd fails and source is lost, decide to justify this as experimental work while learning sufficient javasscript

**STEP 1 just get the code running**

Download an evaluation copy of WebStorm, may as well get help from a good IDE  
Bah need to install Node, do it via a new install of homebrew etc, add npm for good measure.

Write a trivial brandStore.js

Run the code and check it out with some console.logs, oh no ES6 and imports not processed correctly by ES6 parser, discover after an hour or more of googling that I need   
 ‘--experimental-modules’ as a command line option to the Node interpreter and .mjs files

**STEP 2 write a test suite**

OK we are rocking, code runs. Refactor the badly named code.mjs? Nope not till I have a test suite to ensure that the code works the same. Shall I install a test runner? Check out Mocha, or it runs tests serially; that’s poor. And Mocha looks to have a bit of complexity to learn, so given  
I have a lot of my own stuff to work on, so minimising time spent on this task is important, its as easy to use npm to install expect and write a test suite myself.

Best extract out setup of different brandstore contents into a function to keep my tests looking reasonably good, need to therefore add a setter to brandstore.mjs and export it. Not very happy with expressing brandstore contents in JSON and then parsing that, but hey, my time is scant and I’ll learn soon enough to express this in language constructs that are better.

Better test the operation of the brandstore, done. All tests developed using a red green cycle for the standard reason.

Now test ‘getBranding’ in code.mjs (yes in messing around with imports I changed how this is exported in code.mjs). Discover that my array approach in setting up branding allows me to simplify code.mjs thus  
const brands = brandStore // remove .getAll();

OK get a small set of tests for ‘getBranding’ working…. Had to dip into WebStorm’s debugger a bit to step through ‘getBranding’ and get my brandStore data shipshape to make the code run. Particularly this makes me change colours to colors, best to program in American English else there will be no end of trouble.

**STEP 3 a light refactor**

OK lets think is this code hard to read, not really I’m used to terse Ruby syntax. What is difficult?

1. A mix of function types and syntax, some plain functions, some ES6 functions loved by many stack overflow says, at least one anonymous function.
2. Embedded functions, yeah if ES6 works like other languages to control visibility by scope I understand this but ‘validateBodyFont’ not really helping the readability of ‘buildTheme’.
3. ‘getBranding’ has multiple returns each of which involve an invocation of ‘buildTheme’. This looks messy and makes understanding a little harder, I think. I’m not a purist re each function must have only one return, but in this case maybe there is some refactoring to do.

There is a hint for javascript ignoramuses (me) in the question spec, make this more functional programming stylee. Maybe functional programming in JS/ES6 is a way to refactor, better check that out…. But first just for fun fix 2 and 3….

Discover that break doesn’t work in the existing code, so with one google to find out that there is ‘some’ and one edit write this, which passes the tests, bingo!  
  
export function getBranding(user) {

let colors = defaultColors;

let font = defaultBodyFont

if (user.brandId && brandStore.length) {

brandStore.some(function(brand) {

const brandMatch = (brand.id === user.brandId);

if (brandMatch) {

colors = brand.colors;

font = brand.bodyFont;

}

return brandMatch

});

}

return buildTheme(colors, font);

}

Probably this meets the be more of a functional programmer criterion with the anonymous function in ‘some’. In fact there si probably ES6 syntax waiting in the wings for this…  
  
OK refactoring to fix the embedded function added complexity mentioned in 2 is a trivial, move that out, add a parameter ‘bodyFont’, run the tests - bingo again, with ‘buildTheme’ looking cleaner just because of a smaller volume of code.

Now I see it on its own ‘validateBodyFont’ looks a bit overly hard to read… I think I could refactor it using ‘some’ as before, but hey enough for now….

**Summary**

Google, stackoverflow and a Mozilla JS pages are great.

I’m not keen on my test setup, I might make that clearer, and break it out into its own file to simplify the tests.

I could continue refactoring code.mjs, including changing the file name, but you get the idea Craig.

I should have set up a repo on github.