"MRP" technical interview prompt

Key

Anything you are supposed to say in your script is in a blue box. For example, I would read this whole section to the candidate directly
Actions you need to take will be described in green boxes like this
inline guidance about what hints might be appropriate to give or how to elicit “wow” or “doh” moments are given in boxes like this

Reminder

You must make a time stamped note any time you help an applicant in a way that is off script. Helping an applicant can rob them of the opportunity to prove to us how well they work in difficult situations.

Script

Warm Up

During this interview I'll be evaluating your performance based on your ability to communicate your thinking to me, your ability to write clean code that addresses the problem you are trying to solve, and your ability to learn to use new techniques. During the interview I'll be teaching you some, and after you've had a chance to ask questions, you'll be required to use them in a novel way. This reflects what Hack Reactor students are constantly required to do. During this interview we're going to use a style of development called "test driven development", or, TDD. A test, sometimes called an assertion, is a piece of code that we write that tests the behavior of other code we are writing, and gives us a helpful message if it is not behaving in the way we would expect. Test Driven Development means that we will actually write a test for code we are about to write, and then write the code that will make that test pass. We are going to use a function called `assert` to create tests. Take a look at it here.
Copy the following into CoderPad, or, use the CoderPad Question: WARMUP - Assert
`function assert(expectedBehavior, descriptionOfCorrectBehavior) {`

```
  if (!expectedBehavior) {
    console.log(descriptionOfCorrectBehavior);
  } else {
    console.log('test passed');
  }
}
```

Now check out this very rudimentary test created using the `assert` function. The test is currently failing, which means when we run it, we will see the message printed about what should happen to make the test pass.

Copy the following into CoderPad, or, use the CoderPad Question: MRP 1 - Demo Assert

```
var x;

assert(x === 12, "x should be equal to 12");
```

Run the code

Now I'd like you to tell me how we could make this test pass.

Hopefully the applicant understands that they simply have to assign 12 to the variable `x` to make the test pass. If they talk about doing something different, like modifying the `assert` function, or the call to `assert` tell them that you want them to come up with a way to make the test pass without modifying the test itself, or the `assert` function. After they are able to **tell** you how they will make the test pass...

That's right. Go ahead and do that, and then run the code to make sure the test is passing.

Make timestamped note: WARMUP BAR COMPLETE

Main Problem

Okay, great! Now that you understand how testing works, we'll be able to use it throughout the rest of the interview.

As part of the interview, we are going to be comparing arrays. You may not know this, but comparing arrays in JavaScript is different than comparing numbers or strings. We don't have time to go into great detail about this now, but basically, every array, even if they look the same, actually references a unique value in memory and therefore, JavaScript will never say that two arrays compared directly with each other are equal. Check this out.

Paste the following into CoderPad, or, use the CoderPad Question: MRP 2 - Compare Arrays

```
console.log([1, 2, 3] === [1, 2, 3]);
```

Run the code.

So if we want to compare arrays, we need to check that each value at the same index within the two arrays is equal, and, that the arrays are the same size. Here's a function that we can use to help us do just that, along with some assertion tests that describe how it ought to behave. Unfortunately, the function has some bugs in it, as you'll see from running the tests. Please debug the function so that the tests pass.

Paste the following into CoderPad, or, use the CoderPad Question: MRP 3 - testArrayEquality

```
function testArrayEquality(array1, array2) {
  for (var i = 0; i < array1.length(); i++) {
    if (array1[i] !== array2[i]) {
      return false;
    } else {
      return true;
    }
  }
}

var first = [1, 2, 3, 4, 5];
var second = [1, 2, 3, 4, 5];
var third = [1, 9, 2, 4, 6];
var fourth = [1, 2, 3, 4, 5, 6];


assert(testArrayEquality(first, second) === true, 'it should return
true when inputs are equivalent');
assert(testArrayEquality(first, third) === false, 'it should return
false when inputs are different');
assert(testArrayEquality(first, fourth) === false, 'it should
return false when inputs are different');
```

At the end of the interview, you can count the code in `testArrayEquality` towards the applicant's JavaScript Skill Level assessment

Nice work! As it turns out, debugging code that another person has written is a huge part of being a software engineer, and something Hack Reactor students do a ton of.

Now that you know how to write assertions and how to compare arrays, now I have a task for you.

Imagine you are working on a team that is developing manufacturing resource planning software and there are a few things we need to get done today.

Here is a data set of information about the factories we are working with. Please take a moment and review it.

Paste the following into CoderPad, or, use the CoderPad Question: MRP 4 - Factories Data Set

```
/* data set */

var factories = [

  {
    'factory name': 'c&h plush',
    'products': 'sutffed animales',
    'rating': 'B',
    'monthly capacity': 100000,
    'total current orders': 10000,
    'days to deliver': 20,
    'shipping port': ['yantian']
  },
  {
    'factory name': 'Fox Con',
    'products': 'electronics',
    'rating': 'A',
    'monthly capacity': 1000000,
    'total current orders': 1000000,
    'days to deliver': 10,
    'shipping port': ['yantian','shang hai', 'beijing']
  },
  {
    'factory name': 'SS clothing',
    'products':'textiles',
    'rating':'F',
    'monthly capacity': 10000,
    'total current orders': 0,
    'days to deliver': 30,
    'shipping port': ['yantian']
  }

];
```

The product manager on this project has asked that you make sure to use TDD at all phases, so be sure to write failing tests, then write your function so the tests pass.

Your first task is to complete the function `getFactoryCapacities` function that takes in an array of factories and returns an array of tuples with the factory name and the monthly capacity in the tuple. A tuple is an array with a fixed number of values, in this case 2. The following specs should help you write meaningful tests, which will guide your work.

Paste the following into CoderPad, or, use the CoderPad Question: MRP 5 - getFactoryCapacities

Great job! For the next sections, you're going to be doing some custom sorting, so before continuing, I'd like to show you some features of JavaScript's native array sort method. `sort` modifies the original array and returns it. By default, it treats all the elements as strings, and sorts them by Unicode code point value. If you don't know what a Unicode point value is, don't worry, you can just think of it as sorting al

```
/*
Specs:
- it should return an array
- the array should be made up of tuples (arrays with a length of 2)
- the array should have one tuple for each factory
- the first value of the tuple should be a string
- the second value of the tuple should be a number
*/

// EXAMPLE OUTPUT:  [['factory name', monthlyCapacity], ['another
factory name', otherMonthlyCapacity]]

function getFactoryCapacities(factories) {
  // your code here
}
```

Paste the following into CoderPad, or, use the CoderPad Question: MRP 6 - Alphabetical Sort aphabetically.

```
var letters = ['c', 'a', 'b'];
letters.sort();
console.log(letters);
```

Run the code.

If we want to sort by something other than alphabetically, such as by numerical value, we can pass `sort` a custom sorting function. The sorting function we pass in will be used by the `sort` function to compare the elements in the array it is sorting. The sorting function should expect two arguments, and return a number. If the returned number is less than 0, `sort` will place the first argument earlier in the sorted array. If the returned number is greater than 0, it will place the second argument earlier in the sorted array. For this interview, let's just say that if it returns 0, it keeps these elements in the same order.

Here's an example that will help make this clear where we'll use `sort` to sort these numbers in ascending order.

Paste the following into CoderPad, or, use the CoderPad Question: MRP 7 - Ascending Sort

```
var nums = [1, 5, 11, 9, 2, 20];

function sortNums(num1,num2) {
```

```
  if(num1 < num2) {
    // if this function returns a number less than one, `sort` will
sort the first argument, `num1`, as first
    return 1;
  } else {
    // if this function returns a number greater than one, `sort`
will sort the second argument, `num2`, as first
    return -1;
  }
}

nums.sort(sortNums);
console.log(nums);
```

Run the code.

I'm going to ask you to use your own custom sorting function in the next part of the prompt, so before moving on I want to make sure you feel comfortable with the concept. So that I can know whether or not we should spend more time on this example, modify the `sortNums` function above to sort the words in descending order.

Make timestamped note: MINIMUM PROBLEM MAIN BAR COMPLETE

Your next task is to complete the function `getFactoriesSortedByProductionCapacity`. This function takes in the `factories` array and returns an array that contains the factory names sorted by total production capacity. The following specs should help you write meaningful tests, which will guide your work.

Paste the following into CoderPad, or, use the CoderPad Question: MRP 8 - getFactoriesSortedByProductionCapacity

```
/*
Specs:
- it should return an array
- the array should contain 3 elements
- the output array should equal ['Fox Con','c&h plush','SS
clothing']

*/

function getFactoriesSortedByProductionCapacity(factories) {
  //your work here
}
```

Nice! Now implement the following, continuing to use TDD

Paste the following into CoderPad, or, use the CoderPad Question: MRP 9 - Sorting Argument

```
// Adjust your function to accept another argument and return
sorted data based on that other argument.  For example if your
second argument was 'factory rating' return an array sorted by best
factory to worst factory.
```

Nice! Now implement the following, continuing to use TDD

Paste the following into CoderPad, or, use the CoderPad Question: MRP 10 - Ascending Descending

```
// Now change the function again to take a third optional argument
that changes the sorting order from ascending to descending.
```

Awesome! Now use TDD to implement the following:

Paste the following into CoderPad, or, use the CoderPad Question: MRP 11 - availableCapacityInPieces

```
// Write a function that accepts factory names and returns their
available capacity in pcs
```

Sweet! Now use TDD to do the following:

Paste the following into CoderPad, or, use the CoderPad Question: MRP 12 - calculateTime

```
// Given a factory name and N number of pcs, return the amount of
time before that order would be ready.
```