First Array in Memory Step 1

Arrays

In JavaScript, we can group elements in a data structure called array.

An array is an ordered list of elements separated by commas.

Arrays look like this: `[1, 2, 3]`.

Or like this:

```
var famousGrilledCheese = [
  'bread',
  'sharp cheese',
  'butter',
  'tomato',
  'garlic',
  'oregano'
]
```

Arrays are an ordered collection of elements, meaning that the elements in them will always appear in the same order. The array [1, 1, 7], is different from the array [1, 7, 1].

.length Method Arrays have a lot of useful built-in methods. The array `.length` method will return the number of elements in an array:

```
    const studentsNames = ['Kyle', 'Maria', 'Joe', 'Andrew'];

    console.log(studentNames.length);  // 4
```

Accessing Elements In An Array

We can access an element at any time in an array. We just need to call the element by its position in the array. Elements are given a numerical position (index) according to where they are in an array. An array's numerical order always starts at 0, so the first element is in the 0 index, the second in the 1 index, the third in the 2, and so on.

```
    const studentsNames = ['Kyle', 'Maria', 'Joe', 'Andrew'];
                           0       1        2      3
```

To access an element, type the name of the array variable, followed by brackets containing the numerical assignment.

```
const studentsNames = ['Kyle', 'Maria', 'Joe', 'Andrew'];

console.log(studentNames[1]); // 'Maria'
```

To dynamically access the last element in an array, we use the .length method. In our studentsNames array, the length is 4. We know the first element is always going to be 0, and every element after is shifted over one number. So, in our example the last element has an index of 3. Using our length property, we will show how it's done when we don't know the number of element in an array:

```
    const studentsNames = ['Kyle', 'Maria', 'Joe', 'Andrew'];

    console.log(studentNames[studentNames.length - 1]);  // 'Andrew'
```

.push() and .pop() Methods

These methods refer to the adding and removing of elements from the array after its initial declaration.

.push adds an element to the end of the array, incrementing it's length by 1. (.push returns the new length)

```
    const studentsNames = ['Kyle', 'Maria', 'Joe', 'Andrew'];

    studentNames.push('Tini');

    console.log(studentNames);  // ['Kyle', 'Maria', 'Joe', 'Andrew',
'Tini']
```

.pop removes the last element in the array, decrementing its length by 1. (.pop returns the "popped" element)

```
    const studentsNames = ['Kyle', 'Maria', 'Joe', 'Andrew'];

    studentNames.pop();

    console.log(studentNames);  // ['Kyle', 'Maria', 'Joe']
```

.unshift() and .shift() .unshift and .shift are exactly like .push and .pop, except they operate on the first element in the array. .unshift(element) puts a new element in the first position of the array, and .shift() removes the first element in the array.

```
    const studentsNames = ['Kyle', 'Maria', 'Joe', 'Andrew'];

    studentNames.unshift('Renee');
```

```
    console.log(studentNames);  // ['Renee', 'Kyle', 'Maria', 'Joe',
'Andrew']

    studentNames.shift();

    console.log(studentNames);  // ['Kyle', 'Maria', 'Joe', 'Andrew']
```

For Loops

Many times, for loops are used to iterate over all elements in an array. Using the index access technique, we can access each element in an array. To do this, we use the .length method as a stopping point for the loop.

```
    const studentsNames = ['Kyle', 'Maria', 'Joe', 'Andrew'];

    for (let i = 0; i < studentNames.length; i++) {
        console.log(studentNames[i]);
    }

    // 'Kyle'
    // 'Maria'
    // 'Joe'
    // 'Andrew'
```

Additional resources MDN: Arrays MDN: for Loops Task instructions Task 1 Open the file task1.js under the folder first-array. You'll see a group of functions to work on.

There are six functions, read each comment and write the return statements for each of them.

It's a good practice to console log the results to help figure out what's expected.

Task

- Given the following 6 empty functions, go through each comment, and complete the code to return the proper array on each of them.

First Array in Memory - Step 2

Arrays - Task 2

Open the file task2.js under the folder first-array and your code there.

Given an array of strings (words), write a function that would concatenate all the strings in the array to form a sentence. The words should be separated by a space.

Check the task when complete.

Task

- Given an array of strings called words, return a string that has all the words concatenated together.