For Loops

Interating (Looping) Arrays

A loop repeatedly executes a block of code, as long as a condition remains true. There are a few kinds of loops, including "for loops" and "while loops". In this lesson, we will focus on "for loops".

for loops

A for loop considers three pieces of information in parentheses to decide if it will run the code inside its curly braces: a **counter**, a **condition** and an **incrementer**:

```
for (counter, condition, incrementer) {
    // do stuff
}
```

counter

The counter is a number variable that goes up or down each time the loop is run. The counter is usually `i`, but can be any name. The counter is assigned an initial value, typically 0, although that can be any number.

condition

The condition compares the counter to another value. If the condition is true, the code inside the curly braces will run. If the condition is false, the loop ends.

incrementer

The incrementer (or decrementer), specifies the value by which the counter increases or decreases with each iteration of the loop. Typically, a counter will go up by one each time though the loop, as assigned by the shorthand, `i++`.

1. Write a for loop with a counter, `i`, that starts at 0, goes up by one each time, and stops when `i` gets to 10:

```
for (let i = 0; i < 10; i++) {
    console.log(i); // 0, 1, 2...9
}
```

let variables are block scoped

As discussed in previous lessons, a `let` variable is `block scoped`, meaning that it is available only inside the code block in which it is declared.

2. Try to access `i` outside the loop. We get an error:

```
for (let i = 0; i < 10; i++) {
    console.log(i); // 0, 1, 2...9
}

console.log(i); // ERROR: i is not defined
```

The error indicates that `i` does not exist in the global scope, which is fine because `i` was only needed in the loop. Once the loop ended, `i` was deleted from memory ("garbage collected", as they say).

***var* variables are not block scoped**

A `var` is not block scoped, so if you use `var i` for a loop counter, `i` will be instantiated in the global scope--and will still be there after the loop ends. It is said that `var` counters "leak out" of their loop. We don't want variables persisting in memory with nothing to do, so use `let` instead.

3. Do another loop with `var` instead of `let`, and verify that `i` still exists after the loop has ended:

```
for (var i = 0; i < 10; i++) {
    console.log(i); // 0, 1, 2...9
}

console.log('after loop', i); // after loop 10
```

Notice that the value of `i` after the loop ends is 10, which matches the loop condition. Once the counter reached 10, the condition `i < 10` became false, so the loop ended.

4. Change the condition to `i <= 10` to get to 10 inside the loop and 11 outside the loop.

Let's try counting down. Start `i` at 10 and *decrement* by 1 each time with `i--`. When `i` reaches 0, the condition `i > 0` is false, so the loop ends.

5. Write a loop with a counter that decrements (counts backwards):

```
for (let i = 10; i > 0; i--) {
    console.log(i) // runs 10 times
}

console.log(i) // 11
```

The 11 reminds us that we still have a lingering `var i` which "leaked out" of its loop into the global scope.

6. Change `i` to `j` and see that `j` ceases to exist once the loop ends:

```
for (let j = 10; j > 0; j--) {
    console.log(j) // runs 10 times
}
console.log(j) // ERROR: j is not defined
```

infinite loop

An infinite loop is one that never ends, because the condition is always true. The condition is supposed to eventually flip from true to false, but in an infinite loop that never happens due to a flaw in the logic.

This next loop runs forever because `i` starts at 10 and goes up from there. The condition `i > 0` is therefore always true.

7. Write but do not run this infinite loop, as doing so may freeze your browser. Just study it before commenting it out.

```
/* infinite loop; do not run
for (let i = 10; i > 0; i++) {
    console.log(i);
}
*/
```

the += and -= operators

The counter can be incremented / decremented by any value. To increment by 5, it's `i+=5`. To decrement by 2, it's `i-=2`.

8. Run a loop where the counter starts at 0, goes up by 5 each time until it reaches 100 (inclusive):

```
for (let i = 0; i <= 100; i+=5) {
    console.log(i); // 0, 5, 10...95, 100
}
```

continue

The `continue` keyword skips an iteration of a loop. It is used with conditional logic to specify when to skip:

9. Output all numbers from 1-10, with the exception of 7:

```
for(let i = 0; i < 10; i++) {
    if(i == 7) {
        continue; // skip 7
    }
    console.log(i); // 0,1,2,3,4,5,6,8,9
}
```

iterating (looping) arrays

Loops are commonly used to iterate arrays, which means to go through them, item by time. The counter is used to get the current item by index.

In these next steps, we will loop through an array while also working in some review of array methods:

10. Declare an array:

```
    const fruits = ['apple', 'blueberry', 'cherry', 'kiwi', 'lime',
'orange', 'plum'];
```

11. Add a few more items to the end of the array:

```
    fruits.push('apricot');
    fruits.push('papaya')
    fruits.push('grape');
```

12. Add a few items to the beginning of the array:

```
    fruits.unshift('grapefruit');
    fruits.unshift('watermelon');
    fruits.unshift('tangerine');
```

13. Output the array and its length:

```
    console.log(fruits, fruits.length);
    // ['tangerine', 'watermelon', 'grapefruit', 'apple', 'blueberry',
'cherry', 'kiwi', 'lime', 'orange', 'plum', 'apricot', 'papaya', 'grape']
13
```

14. Starting at index 3, splice out 3 items ('apple', 'blueberry', 'cherry'), and replace them with 'lemon' and 'pear':

```
    fruits.splice(3, 3, 'lemon', 'pear');

    console.log(fruits, fruits.length);
    // ['tangerine', 'watermelon', 'grapefruit', 'lemon', 'pear', 'kiwi',
'lime', 'orange', 'plum', 'apricot', 'papaya', 'grape'] 12
```

15. At index 7, insert 4 items without removing any. The new fruits will go in before 'orange':

```
    fruits.splice(7, 0, 'apple', 'blueberry', 'cherry', 'peach');

    console.log(fruits, fruits.length);
    //  ['tangerine', 'watermelon', 'grapefruit', 'lemon', 'pear', 'kiwi',
'lime', 'apple', 'blueberry', 'cherry', 'peach', 'orange','plum',
'apricot', 'papaya', 'grape'] 16
```

16. Iterate the array of 16 items with a for loop. Each time through the loop, make a fruit jellybean.

```
    for(let i = 0; i < 16; i++) {
        let bean = fruits[i] + ' jellybean';
        console.log(bean); // tangerine jellybean, etc.
    }
```

17. Push in a few more fruits, and then sort the array:

```
    fruits.push('banana');
    fruits.push('pineapple');
    fruits.push('mango');
    fruits.sort();

    console.log(fruits, fruits.length);
    // ['apple', 'apricot', 'banana', 'blueberry', 'cherry', 'grape',
'grapefruit', 'kiwi', 'lemon', 'lime', 'mango', 'orange', 'papaya',
'peach', 'pear', 'pineapple', 'plum', 'tangerine', 'watermelon'] 19
```

18. Run the loop again:

```
    for(let i = 0; i < 16; i++) {
       let bean = fruits[i] + ' jellybean';
       console.log(bean);
       // apple jellybean, apricot jellybean, etc.
    }
```

The last three fruits ('plum', 'tangerine', 'watermelon') are not showing up. The problem is, the loop stops when `i=16`, but we now have 19 items.

array.length as loop condition

For a loop that iterates an array, do not use a hard-coded number in the condition. Instead, use the dynamic condition `i < array.length`. This way, the loop fully iterates the array--and then stops.

19. Make a new loop with the dynamic condition `i < fruits.length`:

```
    for(let i = 0; i < fruits.length; i++) {
        let bean = fruits[i] + ' jellybean';
        console.log(bean);
        // apple jellybean, apricot jellybean, etc.
    }
```

string.length

As we recall, the `length` property applies not only to arrays, but to strings, as well.

conditional logic inside a loop

Loops that iterate arrays often include conditional logic to evaluate the individual items. Let's give this a try.

20. Make jellybeans only if the fruit has no more than 5 letters. Btw, we don't really need the bean variable, so just log the string directly:

```
    for(let i = 0; i < fruits.length; i++) {
        if(fruits[i].length <= 5) {
            console.log(fruits[i] + ' jellybean');
            // apple jellybean, grape jellybean, etc.
        }
    }
```

making a new array inside a loop

The above loops are not saving the jellybeans anywhere; the beans are just being "spilled" out onto the console.

21. Do another loop that saves the jellybeans by pushing them into a new array, `jellybeans`, declared above the loop:

```
    const jellybeans = [];

    for(let i = 0; i < fruits.length; i++) {
        if(fruits[i].length <= 5) {
            jellybeans.push(fruits[i] + ' jellybean');
        }
    }

    console.log(jellybeans);
    // ['apple jellybean', 'grape jellybean', 'kiwi jellybean', 'lemon
jellybean, 'lime jellybean', 'mango jellybean', 'peach jellybean', 'pear
jellybean', 'plum jellybean']
```

Let's try some if-else if-else logic in a loop:

- `if` the fruit has a max of 5 letters, it goes into the `lilFruits` array.

- `else if` the fruit has 6-8 letters, it goes into the `medFruits` array.
- `else`, the fruit of 9 or more letters it goes into the `bigFruits` array

22. Declare the three arrays, all empty:

```
    const lilFruits = [];
    const medFruits = [];
    const bigFruits = [];
```

23. Run the loop with the conditional logic. Each part pushes eligible fruits into its respective array:

```
    for(let i = 0; i < fruits.length; i++) {
        if(fruits[i].length <= 5) {
            lilFruits.push(fruits[i]);
        } else if(fruits[i].length <= 8) {
            medFruits.push(fruits[i]);
        } else { // 9+ chars
            bigFruits.push(fruits[i]);
        }
    }

    console.log('lilFruits', lilFruits);
    // ['apple', 'grape', 'kiwi', 'lemon', 'lime', 'mango', 'peach',
'pear', 'plum']
    console.log('medFruits', medFruits);
    // ['apricot', 'banana', 'cherry', 'orange', 'papaya']
    console.log('bigFruits', bigFruits);
    // ['blueberry', 'grapefruit', ' 'pineapple', 'tangerine',
'watermelon']
```

loop with nested if-else

Let's finish the lesson with some nested logic. Building upon what we have so far, if the "big fruit" is a berry, save it to the `berries` array. For this we need a few more berries.

We will use `splice()` to maintain alphabetical order as we add 'boysenberry', 'raspberry' and 'strawberry'. Here is the current array of 19 items for reference:

```
['apple', 'apricot', 'banana', 'blueberry', 'cherry', 'grape',
'grapefruit', 'kiwi', 'lemon', 'lime', 'mango', 'orange', 'papaya',
'peach', 'pear', 'pineapple', 'plum', 'tangerine', 'watermelon']
```

24. We need to put 'boysenberry' needs between 'blueberry' and 'cherry'. Starting at index 4 ('cherry'), remove zero items and add 'boysenberry':

```
    fruits.splice(4, 0, 'boysenberry');
```

splice() with negative index values

The splice method recognizes negative indexes, with the last item at -1. We want to add 'raspberry' and 'strawberry', consecutively, right before 'tangerine', which, as the next to the last item, occupies index -2.

25. Add 'raspberry' and 'strawberry' before 'tangerine':

```
    fruits.splice(-2, 0, 'raspberry', 'strawberry');

    console.log(fruits, fruits.length);
    // ['apple', 'apricot', 'banana', 'blueberry', 'boysenberry',
'cherry', 'grape', 'grapefruit', 'kiwi', 'lemon', 'lime', 'mango',
'orange', 'papaya', 'peach', 'pear', 'pineapple', 'plum', 'raspberry',
'strawberry', 'tangerine', 'watermelon']
```

26. Declare all new empty arrays for a fresh start:

```
    const smFruits = [];
    const mdFruits = [];
    const lgFruits = [];
    const berries = [];
```

27. Do the for loop with nested if-else. To check if a fruit *includes* 'berry', call `includes('berry')` on the item:

```
    for(let i = 0; i < fruits.length; i++) {
        if(fruits[i].length <= 5) {
            smFruits.push(fruits[i]);
        } else if(fruits[i].length <= 8) {
            mdFruits.push(fruits[i]);
        } else { // 9+ chars
            if(fruits[i].includes('berry')) {
                berries.push(fruits[i]);
            } else {
                lgFruits.push(fruits[i]);
            }
        }
    }

    console.log('smFruits', smFruits);
    // ['apple', 'grape', 'kiwi', 'lemon', 'lime', 'mango', 'peach',
'pear', 'plum' ]
    console.log('mdFruits', mdFruits);
    // ['apricot', 'banana', 'cherry', 'orange', 'papaya']
    console.log('lgFruits', lgFruits);
    // ['grapefruit', ' 'pineapple', 'tangerine', 'watermelon']
```

```
    console.log('berries', berries);
    // ['blueberry', 'boysenberry', 'raspberry', 'strawberry']
```

- **END: Lesson 05.01**
- **NEXT: Lab 05.01, Lesson 05.02**