Lesson 05.01

Loops

Interating (Looping) Arrays

A loop executes a block of code, repeatedly, for 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**:

```
```js
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:

```js
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 then comment out and do not run these infinite loops, as doing so may freeze your browser. Just study it before commenting it out.

```
/*
for(let i = 0; i < 10; i--) {
    console.log(i);
}

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
}
```

Challenge: use for loops to produce the following output:

- 0, -3, -6, -9, -12
- 1900, 1920, 1940, 1960, 1980, 2000, 2020

continue

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

9. Starting with 1900, output once each decade (1900, 1910, 1910, 1920, etc.) stopping at 2020, but skipgin 1960:

```
for(let i = 1900; i <= 2020; i=10) {
    if(i == 1960) {
        continue;
    }
    console.log(i); // 1900, 1920, 1940, 1980, 2000, 2020
}
```

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. Push a few items into the end of the array:

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

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

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

13. Output the array and its length:

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

14. Starting with 'cherry', and assuming you don't know the index of any of the items, replace 'cherry' along with the next two items with 'lemon' and 'pear':

```
fruits.splice(fruits.indexOf('cherry'), 3, 'lemon', 'pear');
console.log(fruits, fruits.length); //  ['grapefruit', 'watermelon',
```

```
'tangerine', 'apple', 'blueberry', 'pear', 'lemon', 'orange', 'plum',
'apricot', 'papaya', 'grape'] 12
```

15. Without removing any items, add 'raspberry' and 'mango' right before 'apricot':

```
fruits.splice(fruits.indexOf('apricot'), 0, 'raspberry', 'mango');

console.log(fruits, fruits.length);
```

// ['grapefruit', 'watermelon', 'tangerine', 'apple', 'blueberry', 'pear', 'lemon', 'orange', 'plum', 'raspberry', 'mango', 'apricot', 'papaya', 'grape'] 14 ```

16. Iterate array with a for loop.

- Each time through the loop, make a jellybean.
- Number the output from 1-14

```
for(let i = 0; i < 14; i++) {
    let bean = `${i+1}. ${fruits[i]} jellybean`;
    console.log(bean);
}
// grapefruit jellybean
// .. etc. ..
// tangerine jellybean, etc.
```

17. Push in three more fruits; then sort and output the array:

```
fruits.push('banana', 'pineapple', 'peach');
console.log(fruits, fruits.length);
// ['apple', 'apricot', 'banana', 'blueberry'
// ..etc. .. 'plum', 'raspberry', 'tangerine', 'watermelon'] 17
```

18. Run the loop again:

```
for(let i = 0; i < 14; i++) {
   let bean = `${i+1}. ${fruits[i]} jellybean`;
    console.log(bean);
}
// apple jellybean
// ... etc. ...
// plum jellybean
```

The last three fruits ('raspberry', 'tangerine', 'watermelon') weren't outputted, because we are running the loop only 14 times for 17 array items.

array.length for loop condition

To make a loop dynamically respond to changes in the size of the array, don't hard-code the number of iterations. Use array.length, instead.

19. Change the loop condition so that it will always run as many times as there are items in the array. Dispense with the bean variable and the item numbering, and just log the jellybean directly:

```
    for(let i = 0; i < fruits.length; i++) {
    console.log(`${fruits[i]} jellybean`);
    /* apple jellybean
    ... etc. ...
    watermelon jellybean */
}
```

Now, we get all 17 jellybeans.

loops with conditional logic

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

Let's make jellybeans again, but this time only if the fruit has five or fewer characters. To check how many letters the fruit is, use the string.length property.

20. Make jellybeans, but only if the fruit is five letters or less:

```
for(let i = 0; i < fruits.length; i++) {
    if(fruits[i].length <= 5) {
        console.log(`${fruits[i]} jellybean`);
    }
} // jellybeans: 'apple', 'grape', 'lemon', 'mango', 'peach', 'pear',
'plum'
```

Challenge:

21. Make different fruit treats, depending on the number of letters

- if the fruit is 5 or fewer letters, make jellybeans
- if the fruit has 6-8 letters, make popsicles: "orange popsicle"
- if the fruit is 9 or more letters, make lollipops: "tangerine lollipop"

```
console.log("\nMake jellybeans, popsicles or lollipops:\n");
```

```
for(let i = 0; i < fruits.length; i++) {
    if(fruits[i].length <= 5) {
        console.log(`${fruits[i]} jellybean`);
    } else if(fruits[i].length <= 8) {
        console.log(`${fruits[i]} popsicle`);
    } else {
        console.log(`${fruits[i]} lollipop`);
    }
}
// apple jellybean
// apricot popsicle
// ... etc. ...
// watermelon lollipop
```

making a new array while looping

The loop is not saving the treats anywhere; they're just being dumped out onto the console.

22. Refactor the loop so that treats are saved to a new array, which we declared before the above the loop:

```
console.log("\nSave jellybeans, popsices and lollipops to the treats
array:\n");

const treats = [];
for(let i = 0; i < fruits.length; i++) {
    if(fruits[i].length <= 5) {
        treats.push(`${fruits[i]} jellybean`);
    } else if(fruits[i].length <= 8) {
        treats.push(`${fruits[i]} popsicle`);
    } else {
        treats.push(`${fruits[i]} lollipop`);
    }
}
console.log(treats); // ['apple jellybean', 'apricot popsicle' .. etc.
.. 'watermelon lollipop']
```

23. Push in three more berries and then sort the array:

```
fruits.push('strawberry', 'blackberry', 'boysenberry');
fruits.sort();
```

Challenge:

24. Refactor the fruit loop, so that we still make treats, based on the length of the word. But if the fruit is a 'berry', make jam instead of lollipops. Also this time, since we are using fruits[i] so often, simplify fruits[i] by saving it to a variable:

```
const treats2 = [];
for(let i = 0; i < fruits.length; i++) {
    let fruit = fruits[i];
    if(fruit.includes("berry")) {
        treats2.push(`${fruit} jam`);
    } else if(fruit.length < 6) {
        treats2.push(`${fruit} jellybean`);
    } else if(fruit.length < 9) {
        treats2.push(`${fruit} popsicle`);
    } else { // 9+ letters, but not a berry
        treats2.push(`${fruit} lollipop`);
    }
}
console.log('treats2:', treats2);
```

making arrays of objects with a loop

It can be very useful to make arrays of objects on a loop. As objects, the resulting array items can have as many properties as you like:

25. Make a loop that goes through this array of full names, and makes objects consisting of three properites each: firstName, lastName, pin, the last being a random 4-digit PIN, with leading 0's, as needed.

```
const basketballStarsArr = ["LeBron James", "Micheal Jordan", "Larry
Bird", "Julius Erving", "Wilt Chamberlain"];
// Push each object into this given array:
const basketballStarObjArr = [];

for(let i = 0; i < basketballStarsArr.length; i++) {

    // 26. Split the current array item, the full name, into an array
of two names
    let namesArr = basketballStarsArr[i].split(" ");

    // 27. Save the first name, which is the first array item, to a
variable
    let fName = namesArr[0];

    // 28. Save the last name, which is the second array item, to a
variable
    let lName = namesArr[1];

    // 29. Generate a random integer from 0-9999
    let pin = Math.floor(Math.random()*10000);

    // 30. Precede the number with leading 0's, as needed
    if(pin == 0) {
        pin = '0000';
    } else if(pin < 10) {
```

```
            pin = '000' + pin;
        } else if(pin < 100) {
            pin = '00' + pin;
        } else if(pin < 1000) {
            pin = '0' + pin;
        // it's already 4 digits, but stringify it
        } else {
            pin = '' + pin;
        }

        // 31. Make an object containing the three properties
        let obj = {
            firstName: fName,
            lastName: lName,
            pin: pin
        };

        // 32. Push the object into the array
        basketballStarObjArr.push(obj);

    }

    // 33. Log the newly made array of objects
    console.log(basketballStarObjArr);
```

26. Given this array of names, make passwords consisting or: the first name, backwards, all lowercase

- a special character: "#" if the first name has 4 letters or less "&" if the first name has exactly 5 letters "%" if the first name has more that 5 letters
- the first three letters of the last name
- a piece of punctuation: an exclamation point ("!") if the first name and last name have the same number of letters a question mark ("?") if the first name is longer than the last name an colon ("😊 if the last name is longer than the first name
- the number of letters in the full name
- if the first and last name start with the same letter: add an equal sign "=" to the end otherwise, add an asterisk "*"
- finally, if the last name starts with a vowel: add "V" for vowel otherwise add "C" for consonant--unless the last name starts with "Y", in which case, add "X"

Save the passwords to an array of objects that include the names, divided into first and last name

```
```js
const ballplayersArr = ["Hank Aaron", "Ernie Banks", "Carl Yastrzemski",
"Mickey Mantle", "Tony Oliva", "Babe Ruth", "Willie Mays", "Nolan Ryan"];

const ballplayersObjArr = [];

for(let i = 0; i < ballplayersArr.length; i++) {
```
```

```
    // 35. Split the first and last name into array
    let names = ballplayersArr[i].split(" ");

    // 36. Assign the first and last names to variables
    let fName = names[0];
    let lName = names[1];

    // 37. Split the first name string into an array
    let fNameCharsArr = fName.split("");

    // 38. Reverse the array
    let fNameCharsArrRev = fNameCharsArr.reverse();
    // make a backwards string from the reversed array
    let fNameBackwards = fNameCharsArrRev.join("");

    // 39. O, chain split(), reverse() and join() together:
    let fNameRev = fName.split("").reverse().join("");

    // 40 Start concatenating the password, beginning with the first name
backwards:
    let pswd = fNameRev;

    // 41. Add a special character, based on the length of the first name:
    // add "#", if the first name has 4 letters or less
    if(fName.lenght <= 4) {
        pswd += "#";
    // add "&", if the first name has exactly 5 letters
    } else if(fName.lenght == 5) {
        pswd += "&";
    } else { // first name has 5+ letters, so add "%""
        pswd += "%";
    }

    // 42. Add the first three letters of the last name:
    pswd += lName.slice(0,3);

    // 43. If first name and last name have the same number of letters,
add "!"
    if(fName.length == lName.length) {
        pswd += "!";
    // if first name is longer than last name, add "?"
    } else if(fName.length > lName.length) {
        pswd += "?";
    } else { // last name is longer than first; add ":"
        pswd += ":";
    }

    // 44. Check if the first and last names start with the same letter
    if(fName[0] == lName[0]) {
        pswd += "=";
    } else {
```

```
        pswd += "*";
    }

    // 45. Check if the first name starts with a vowel, consonant or "y"
    if("aeiou".includes(lName[0])) {
        pswd += "V";
    } else if(lName[0] == "y") {
        pswd += "X";
    } else {
        pswd += "C";
    }

    // 46. Make an object of 3 properties:
    let obj = {
        firstName: fName,
        lastName: lName,
        password: pswd
    };

    // 47. Add the object to the new array
    ballplayersObjArr.push(obj);
}

// 48. Output the new array
console.log(ballplayersObjArr);

// Expected output:
/* namesAndPasswords = [
    {firstName: "Hank", lastName: "Aaron", password: "knah#Aar:9*V"},
    {firstName: "Ernie", lastName: "Banks", password: "einre&Ban!10C="},
    {firstName: "Carl", lastName: "Yastrzemski", password: "lrac"},
    -- etc. --
]; */
```