UNIT 06

LESSON 06.05

JavaScript BOOTCAMP

Sorting Arrays of Objects by Key

Randomizing (Shuffling) Arrays

Object.keys(), Object.value()

In this lesson, we will again work with objects and arrays as we learn how to:

- A. sort arrays of strings vs. arrays of numbers
- B. combine multiple arrays into a single array of objects
- C. sort arrays of objects by number key
- D. sort arrays of objects by string key
- E. randomize (shuffle) arrays of objects
- F. make an array from an object's keys
- G. make an array from an object's values

1. Start by declarings 3 arrays, each with 7 items.

```
    const fNames = ['Pat', 'Jose', 'Ping', 'Amy', 'Shujia', 'Jed',
'Betty'];
    const lNames = ['Hanson', 'Diaz', 'Liu', 'Smith', 'Guo', 'Martin',
'Jones'];
    const ids = [7445, 1214, 748, 4545, 962, 53, 6342];
```

A. sort arrays of strings vs. arrays of numbers

2. Let's review sorting arrays of strings, which we first looked at in Unit 4:

```
    lNames.sort();
    console.log(lNames);
    // ['Diaz', 'Guo', 'Hanson', 'Jones', 'Liu', 'Martin', 'Smith']
```

3. What we haven't tried yet is to sort arrays of numbers. Let's try that now (it won't work):

```
    ids.sort();
    console.log(ids);
    // [1214, 4545, 53, 6342, 7445, 748, 962]
```

With no arguments, the **sort()** method only works on strings. It even "stringifies" numbers, seeing the digits as characters. From that point of view, 9 is greater than 1, just as "z" is greater than "a".

To sort arrays of numbers, we pass a **callback** to the **sort()** method:

- a **callback** is a function passed to another function as its argument
- the callback function is **anonymous** and has two parameters: **a** and **b**
- **a** and **b** as arguments are consecutive items in the array
- the function returns **a - b**, which reverses the position of consecutive items as needed to sort them. This happens under the hood on a loop.

4. Call the sort method on the **ids** array; pass in the callback function:

```
ids.sort(function(a, b) {
    return a - b;
});
console.log(ids);
// [53, 748, 962, 1214, 4545, 6342, 7445]
```

B. combine multiple arrays into a single array of objects

The items in the 3 arrays (**fNames, lNames, ids**) correspond to each other, so the first item of each array line up to give us Pat Hanson in Sales, with an id of 7445. This means that we can -- and should -- consolidate the data into one array of objects.

5. Declare a new empty array, **team**, and loop the length of one of the 4 arrays to be combined:

```
const team = [];
for(let i = 0; i < ids.length; i++) {
}
```

6. Declare an object, and give it four properties. The key-value pairs are derived from the corresponding arrays. Push the completed object into the array:

```
const team = [];
for(let i = 0; i < ids.length; i++) {
    let obj = { fName: fNames[i], lName: lNames[i], id: ids[i] };
    team.push(obj);
}
```

7. Below the loop, log the array along with a sampling of its data:

```
console.log(team);
/*
0: {fName: 'Pat', lName: 'Hanson', id: 7445}
```

```
    1: {fName: 'Jose', lName: 'Diaz', id: 1214}
       -- ETC. --
    6: {fName: 'Betty', lName: 'Jones', id: 6342}
    */
    console.log(team[0]);
    // {fName: 'Pat', lName: 'Hanson', id: 7445}
    console.log(team[1].fName); // Jose
    console.log(team[2].lName); // Liu
    console.log(team[3].id); // 4545
```

C. sorting arrays of objects by number key

8. Call the **sort()** method on the **team** array. It does nothing, because the array items are not strings or numbers, but objects:

```
    team.sort();
    console.log(team.sort()); // no change
```

To sort by an object by **key**, we need to pass in the **callback function** with the **a** and **b** arguments. To sort by numeric key return **a.key - b.key**:

9. Calling the **sort()** method on the **team** array again, pass it the callback with arguments **a** and **b**:

```
    team.sort(function(a, b) {
    });
```

10. Sort by **id** by returning **a.id - b.id**:

```
    team.sort(function(a, b) {
        return a.id - b.id;
    });
```

11. Log **team** again. The objects are sorted by **id**, ascending:

```
    console.log(team);
    /* 0: {fName: 'Jed', lName: 'Martin', id: 53}
        -- ETC --
    6: {fName: 'Pat', lName: 'Hanson', id: 7445} */
```

12. To sort in descending order call the **reverse()** method:

```
    team.reverse();
    console.log(team);
```

13. Or sort in descending order by chaining **reverse()** onto **sort()**:

```
    team.sort(function(a, b) {
        return a.id - b.id;
    }).reverse();
    console.log(team);
```

D. sorting arrays of objects by string key

Sorting objects by keys with string values instead of numbers requires different logic. Here **a** and **b** are compared to see which is greater. The return value is 1 or -1, depending on the result of the evaluation.

14. Sort the team array by last name:

```
    team.sort(function(a, b) {
        if(a.lName > b.lName) {
            return 1;
        } else {
            return -1;
        }
    });
    console.log(team);
```

15. Sort by first name, *descending*, by chaining **sort()** into **reverse()**:

```
    team.sort(function(a, b) {
        if(a.fName > b.fName) {
            return 1;
        } else {
            return -1;
        }
    }).reverse();
    console.log(team);
```

E. randomize (shuffle) arrays of objects

Another use of the **sort(callback)** method is to randomize an array. Here the return value is **return 0.5 - Math.random()**, which has a 50-50 chance of being positive or negative.

16. Declare an array of consecutive numbers:

```
    let nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
```

17. Randomize the numbers:

```
    nums.sort(function(a, b) {
        return 0.5 - Math.random();
    });
    console.log(nums); // [12,6,4,2,9,8,15,1,7,13,5,11,10,14,3]
```

The random sort / shuffle works with strings, too.

18. Declare an array of alphabetized strings and then randomize it:

```
    let froots = ['apple', 'banana', 'cherry', 'grape', 'kiwi', 'mango', 
'orange', 'papaya', 'peach', 'pear', 'tangerine'];

    froots.sort(function(a, b) {
        return 0.5 - Math.random();
    });

    console.log(froots); // ['kiwi', 'grape', 'cherry', 'pear', 'mango', 
'banana', 'peach', 'orange', 'tangerine', 'apple', 'papaya']
```

F. make an array from an object's keys

Object.keys()

The **Object.keys()** method returns an array of the object's values. The **team** array has 7 objects, all with the same 3 keys: **fName, lName** and **id**.

19. Save the first object in the array as its own variable and then save the **keys** to a new aray:

```
    const obj = team[0];
    const keysArr = Object.keys(obj);
    console.log(keysArr); // [fName, lName, id]
```

G. make an array from an object's values

Object.values()

The **Object.values()** method returns an array of the object's values.

20. Save the object **values** to a new aray:

```
    const valuesArr = Object.values(obj);
    console.log(valuesArr); // ['Pat', 'Hanson', 7445]
```

21. Use **pop()** to remove the last item, the id:

```
    valuesArr.pop();
    console.log(valuesArr); // ['Pat', 'Hanson']
```

22. Use **join()** to turn the **valuesArr** array into a string of first and last name:

```
    const fullName = valuesArr.join(' ');
    console.log('Greetings, ' + fullName + '!');
    // Greetings, Pat Hanson
```

- **END: Lesson 06.05**
- **NEXT: Lesson 06.06**