Arrays in JavaScript

Software Development Bootcamp

Iteration, and Iteration Methods

Topic

Iteration

What Is Iteration?

- Iterating over an array means going through each item in the array one by one.
- Think of it like checking off each name on a list to make sure you've seen them all.
- JavaScript has several ways to do this easily

for loop

- Here we log the name of each fruit in the console. The “i” in `fruits[i]` represents each iteration.

```
let fruits = ['apple', 'banana',
'cherry']
// for loop for iteration
for (let i = 0; i <
fruits.length; i++) {
    console.log(fruits[i])
}
```

for...of

- `let candy` creates a variable to hold each item
- `of candies` defines the array being iterated over

```
let candies = ['Snickers', 'Skittles', 'M&Ms']
// iterating over the candies array using a for...of loop
for (let candy of candies) {
   console.log(candy)
}
```

Topic

Iteration Methods

Iteration Methods

- JavaScript provides built-in methods to perform actions on each element of an array.
- These methods offer more concise and expressive ways to work with arrays.

Map()

- Creates a new array by transforming each element
- Doesn't modify the original array

```
let ages = [12, 13, 14, 15];
// Create a variable newAges whose value is the new array
let newAges =
ages.map(function(age) {
   return age + 1
})
// newAges is an array [13, 14, 15, 16]
```

forEach()

- Executes a function for each array element
- Doesn't create a new array

```
let candies = ['Snickers', 'Twizzlers', 'M&Ms']
// Calling the forEach method on the candies array
candies.forEach((candy) => {
// forEach element in the array perform this action
   console.log(`mmm...${candy}`)
})
```

map() & forEach() Key Differences

- **Return Value**:
  - `map()` returns a new array
  - `forEach()` returns undefined
- **Array Modification**:
  - `map()` creates a new array without changing the original.
  - `forEach()` can modify the original array (if desired)
- **Chaining**:
  - `map()` can be chained with other array methods
  - `forEach()` cannot be chained (returns undefined)
- **Use Cases:**
  - Use `map()` when you need a new array based on the original
  - Use `forEach()` when you just need to perform an action on an element

Reduce()

- Reduces an array to a single value
- Useful for calculations on array data

```
let scores = [10, 20, 30, 40];
// Calling the reduce function on the
scores array
let totalScore = scores.reduce(function
(total, score) {
 return total + score;
}, 0);
console.log(totalScore);
// totalScore is 100
```

Filter()

- Creates a new array with elements that pass a test
- Original array remains unchanged

```
let animals = ["dog", "cat",
"elephant", "bird", "dolphin"];
// Creating a new variable and
calling the filter method on the
animals array
let threeLetterAnimals =
animals.filter(function (animal) {
 return animal.length === 3;
});
// threeLetterAnimals is a new
array ['cat', 'dog']
```

Find()

- Returns the first element that satisfies a condition
- Stops searching after finding a match

```
let fruits = ["apple", "banana",
"cherry", "date"];
// Creating a new variable and
calling the find method on the
fruits array
let fruitWithC =
fruits.find(function (fruit) {
 return fruit.charAt(0) === "c";
});
// fruitWithC is the string
"cherry"
```

Concise Inline Function

- Shorter way to write functions
- Uses arrow => syntax
- Improves readability

```
let numbers = [5, 12, 8, 21, 6];
// Creating a variable result and calling the find function on the numbers array, and immediately returning the new values
let result = numbers.find(num => num > 10);
```

Method Chaining

- Link multiple array methods together
- Performs operations in sequence
- Enhances code readability and efficiency

```
let numbers = [1, 2, 3, 4, 5];
// Double each number and find all
number greater than 5
let result = numbers.map(num => num
* 2).filter(num => num > 5);
```

Why Use Iteration Methods?

- **Readability**: Make code more expressive and easier to understand. Clearly communicate intent of operations on arrays
- **Versatility**: Provide solutions for common array operations. Can be combined for complex data transformations
- **Error Reduction**: Reduce chances of off-by-one errors common in indexed loops. Handle edge cases (like empty arrays) gracefully

Exercise

Applying Array Iteration Methods