Course Link

Course: "JavaScript: Arrays" by Jamie Pittman!

Notes prepared by: Gotur Shrinivasa

Topics Covered:

Array: properties and methods:

---

1)**length**
2)foreach()
3)sort()
4)reverse()
5)find()
6)findIndex()
7)include()
8)every() and some()
9)push() and pop()
10)shift and unshift()

1.Array Basics

array.length: length property gives the length of the array and it is a number.

Direct Assignment: This will override the element in an array.

```
const animals= ["cat","dog","lion","tiger"];
pets[0] = "elephant";
console.log(pets);  //elephant, dog, lion, tiger
```

foreach(): foreach method executes for a function once for each element in an array.

→ foreach **does not mutate the original array**.

→ the function executed on each element in an array is a callback function.

function(currentValue, index, array)

→ foreach also accepts thisArg as an argument, this is optional.

array.forEach(function(currentValue,index,array), thisArg)

Example on For loop:

```
const fruits = ["apple", "banana","orange"]
    for(var i=0; i<fruits.length; i++) {
        console.log(fruits[i]);
    }
```

→ for a for loop, we need to set a length of an array

→ for a foreach loop, it will be taken automatically

Example on Foreach loop:

```
const fruits = [“apple”,”banana”,”orange”]
fruits.foreach((fruit) => {console.log(fruit)});
```

Example2:

const scores = [40,50,60,70,80,90]

→ write a function which will print the scores

→ call the function only when the score is more than 65.

→ use foreach function to iterate over each element in an array to check if the element > 65 or not.

solution:

```
const scores = [40,50,60,70,80,90]

const printScore = (score) => console.log(‘score over 65’, score);

score.foreach((score) => {
if(score > 65) {
    printScore(score);
  }
});
```

2.Working with order data

→ sort(): By default an array is sorted in ascending order

Eg: const pets = [“cat”,”dog”,”mouse”]

console.log(pets.sort()); //[“dog”,”cat”,”mouse”];

→ compare(): compare function can be written in one or two ways.

→explicitly written a function expression:

array.sort(function(a, b) { return a-b });

→ written with es6 syntax:

→ array.sort((a,b) => { return a-b });

The compare function can alter the sort order.

The function can return positive or negative or zero.

The values are sorted based on the return value.

Example: array.sort((a, b) => { return a-b });

a -b = positive —-> b sorted before a
a-b = negative —-> a sorted before b
a-b = 0 ——————-> items stay in the same order

Example: How to sort an array alphabetical order:

const animals = [“mouse”, “lion”,”Tiger”,”cheetah”, “Leopard”]

console.log(animals.sort(animals));

sort an array of numbers in ascending order.

const numbers = [94, 99, 70, 80, 40]
numbers.sort((a,b) => a-b);
console.log(numbers);

sort an array of numbers in descending order.

const numbers = [94, 99, 70, 80, 40]
numbers.sort((a,b) => b-a)
console.log(numbers);

→ reverse(): original array will get changed if we apply a reverse method

Eg1: const scientists = [“albert”, “newton”, “CV Raman”, “Marie curie”];
console.log(scientists.reverse());
//[“Marie curie”,”CV Raman”,”newton”,”albert”]

Eg2: avoid mutating original array
const scientists = [“albert”, “newton”, “CV Raman”, “Marie curie”];
const copyScientists = [...scientists]; //using spread operator copied array
copyScientists.reverse();
console.log(copyScientists);// [“Marie curie”, “CV Raman”, “newton”,”albert”]
console.log(scientists);// [“albert”, “newton”, “CV Raman”, “Marie curie”];

→ find() and findIndex(): Returns the first element in an array or index that possesses the testing function.

find(): First value matched will get returned, if nothing matched then undefined will get returned.

findIndex(): Returns the first index that possesses the testing function, if there is no match then returns -1.
find() and findIndex() will not mutate the original array.

Eg: Find the first grade that is less than 80
const grades = [99, 83, 90, 75, 65];

solution: const underEighty = grades.find((grade) => grade < 80);
console.log(underEighty); // 75

Eg: Find the index of the first grade that is over eighty
const grades = [99, 83, 90, 75, 65];
const overEightyIndex = grades.findIndex((grade) => grade>80); // 0

Challenge

//Sort, reverse, find, findIndex

```
const students = [
                    { name: 'John', grade: 75 },
                    { name: 'Jane', grade: 93 },
                    { name: 'samanta', grade: 90 },
                    { name: 'Michael', grade: 94 }
                ]
```

Question1:
Sort the array of students based on their grade in descending order(largest to smallest).

```
students.sort((a, b) => b.grade - a.grade);
output:
[
  { name: 'Michael', grade: 94 },
  { name: 'Jane', grade: 93 },
  { name: 'samanta', grade: 90 },
  { name: 'John', grade: 75 }
]
```

Question2: After sorting, reverse the order of the array.

```
students.reverse();
console.log(students);
output:
[
  { name: 'John', grade: 75 },
  { name: 'samanta', grade: 90 },
  { name: 'Jane', grade: 93 },
  { name: 'Michael', grade: 94 }
]
```

Question3: Find a student in an array, who has a grade over 90.

```
const getStudent = students.find((student) => student > 90);
console.log(getStudent) //{name: “Jane”, grade: 93};
```

3.Evaluating data for a single value

→includes(): Returns a boolean value if the array includes a specific value.

array.includes(value, fromIndex)

Value is case sensitive.

```
const paymentApps = [“Phonepe”,”GPay”,”Stripe”,”Bharatpe”];
const isPhonepeExist = paymentApps.includes(“Phonepe”);
console.log(isPhonepeExist); // true
```

→some() : Does the array contain some elements that pass a test?

Returns boolean

It returns true if atleast one element matches the criteria

array.some(function(element, index, array), this.arg)

→every(): Does every element in an array pass the test?

Returns boolean

It returns true only if all the elements matches the criteria

array.every(function(element, index, array), this.arg)

Eg1: const temps = [88, 89, 90, 91, 92, 70];

```
const some = temps.some(temp => temp >90); // true
const every = temps.every(temp => temp>60)// true
```

Eg2: const states = [“California”, “New york”,”New jersey”,”Alaska”]

Do some states have “New” in their name?

```
const someStates = states.some((state) => state.startsWith(‘New’));
console.log(someStates); // true
```

Do every state have “New” in their name?

```
const everyState = states.every((state) => state.startsWith(‘New’));
console.log(everyState); // false
```

Challenge

const bowlingScores = [154, 204, 300, 184, 286];

Challenge1: Does the array of bowling scores includes 300

```
const isScorePresent = bowlingScores.includes(300);
console.log(isScorePresent);
```

Challenge2: Are some of the scores under 150?

```
const isUnder = bowlingScores.some((score) => score< 150); // false
```

Challenge3: Is every score an even number?

```
const isScoresEven = bowlingScores.every((score) => score %2 === 0);
console.log(isScoresEven);
```

4.Implementing stacks and queues

stack is a data structure that holds a list of items and operates using **Last In, First out**
Both of these methods change the length of the array and the content of the array.

→**push():** Push is used to add the element/elements of the array to its end.

array.push(element(s))

Eg: const juices = [“apple”,”pineapple”,”orange”,”banana”];

```
juices.push(“grapes”);
console.log(juices); //[“apple”,”pineapple”,”orange”,”banana”];
```

→**Pop():** Pop is used to pop out or remove the element of the array from its end.

array.pop()

pop does not accept any parameter.

Eg: const juices = [“apple”,”pineapple”,”orange”,”banana”];

```
const poppedValue = juices.pop();// “banana”
console.log(juices); // [“apple”,”pineapple”,”orange”];
```

Queue is a data structure that holds a list of items and operates using **First In, FirstOut**

→**unshift():** unshift is used to add the elements to the beginning of the array.

```
array.unshift(element(s));
const games = [“cricket”,”basketball”,”baseball”];
games.unshift(“Tennis”);
console.log(games); //[“Tennis”,”cricket”,”basketball”,”baseball”]
```

→**shift():** shift is used to remove the elements from the beginning of the array.

```
array.shift();
const games = [“cricket”,”basketball”,”baseball”];
const shiftedGame = games.shift();
console.log(shiftedGame); //”cricket”
console.log(games); // [”basketball”,”baseball”];
```

Challenge

```
const foods = [
{ food: 'raspberries', type: 'fruit'},
{ food: 'orange', type: 'fruit' },
{ food: 'broccoli', type: 'vegetable'},
{ food: 'quinoa', type: 'grain'}
];
const blackBeans = { food: 'black beans', type: ' legume'};
const chiaSeeds = { food: 'chia seeds', type: 'seed'};
```

//Challenge Question #1

Remove the last item in the foods array.

```
foods.pop();
console.log(foods);
```

//Challenge Question #2

Remove the first item in the foods array.

```
foods.shift();
console.log(foods);
```

//Challenge Question #3

Add the variable blackBeans to the beginning of the foods array.

```
foods.unshift(blackBeans);
console.log(foods);
```

//Challenge Question #4

Add the variable chiaSeeds to the end of the array.

```
foods.push(chiaSeeds)
console.log(foods)
```

//What does your final array look like?

```
foods = [
{ food: 'black beans', type: ' legume'},
{ food: 'orange', type: 'fruit' },
{ food: 'broccoli', type: 'vegetable'},
{ food: 'chia seeds', type: 'seed'}
]
```

—----------

5.Advanced Methods

→**map:** map method creates a new array based on the function applied to each element in the array you're iterating over.

```
let new= array.map( function(currentValue, index, array), thisArg );
Eg: const marks = [90, 92, 88, 96];
    const graceMarks = marks.map(mark => mark+4);
        or
     const graceMarks = marks.map((mark) => {
return mark+4;
})
```

```
Eg: const friends = [{ firstName: "Shrinivasa", lastName: "Gotur"},
                     { firstName: "Robert", lastName: "Junior" },
                     { firstName: "Sachin", lastName: "Tendulkar" }]
```

Create a new array which has only full names

```
const fullNames = friends.map((friend) => `${friend.firstName}
${friend.lastName}`);
console.log(fullNames);
```

→**filter:** creates a new array based on whether or not elements pass the test provided by the function.
let new= **array.filter(function(currentValue, index, array), thisArg)**;
Eg: const students = [{ name: "Ramesh", marks: 90 },
{ name: "Suresh", marks: 95 },
{ name: "Paramesh", marks: 92 },
{ name: "Pratik", marks: 82 }
]
const distinction = students.filter((student) => student.marks > 85);
console.log(distinction);

→**Reduce:** **Reducing an array to a single value.**
It executes reducer function against each item in an array returning a single value.
array.reduce(function(accumulator, currentValue, index, array), intialValue);
• accumulator (acc): required, accumulated value or total, which is returned value
• currentValue(el): required, element being processed in
the array
• index: optional, index of the currentValue
• array: optional, original array being iterated over
• initial Value: optional, value to use for the initial accumulator;
if nothing is passed, the first value of array is used

const numbers = [10,20,30];
const addition = numbers.reduce((add, number) => add+number);
console.log(addition);

→**FlatMap():** the flat map method returns a new array by calling a callback function to each item of the array and then flattering it one level.
It first applies map() and then flat().
let new= **array.flatmap(function(currentValue, index, array), thisArg)**;
Eg1:
const allowance = [[20], [15], [18]];
const doubleAllowance = allowance. flatMap((value) => [value * 2]); // [40, 30,36]

Eg2:const wordJumble = ['shampoo' ,'conditioner', 'soap'];
/You want to create a word jumble which requires you to split
//the following array of words into individual letters. You want
/a new array, that is flattened to a single level.

```
//Hint: You will need to use the string method „split ()
const splitWords = wordJumble. flatMap ((word) => word. split(‘ ’));
console. log (splitWords);
//output: ['s', 'h', 'a', 'm', 'p', 'o', 'o', 'c', 'o', 'n', 'd', 'i', 't', 'i', 'o', 'n', 'e', 'r', 's', 'o', 'a', 'p']
```