JavaScript: Looping

The Complete Web Developer in 2019

The Complete Web Developer in 2019
Zero to Mastery
Andrei Neagoie
Lecture Notes by Stephanie

Javascript Looping

for
while
do
forEach (new in ECMAScript 5)

Advanced JavaScript:

map
filter
reduce

textbook:
https://eloquentjavascript.net/02_program_structure.html

```
> var toDoList = [
      "clean room",
      "brush teeth",
      "exercise",
      "study js",
      "eat healthy"
  ];
```

```
  toDoList.length
< 5
```

For Loop (for loop is a type of while loop including counter)

```
  for (var i=0; i < toDoList.length; i++) {
      toDoList[i] = toDoList[i] + "!";
  }
```

```
> toDoList
< ▶ (5) ["clean room!", "brush teeth!", "exercise!", "study
    js!", "eat healthy!"]
```

For Loop example 2

```
> var toDoList = [
      "clean room",
      "brush teeth",
      "exercise",
      "study js",
      "eat healthy"
  ];

  for (var i=0; i < toDoList.length; i++) {
      toDoList.pop();
  }
```

```
> toDoList
<- ▶ (2) ["clean room", "brush teeth"]
```

For Loop example 3

```
> var toDoList = [
      "clean room",
      "brush teeth",
      "exercise",
      "study js",
      "eat healthy"
  ];

  var toDoListLength = toDoList.length;
  for (var i=0; i < toDoListLength; i++) {
      toDoList.pop();
  }
```

```
> toDoList
<- ▶ []
```

While Loop

```
> var counterOne = 0;
  while (counterOne < 10) {
      console.log(counterOne);
      counterOne++
  }
```

```
0                                   pathturbo.js:1
1                                   pathturbo.js:1
2                                   pathturbo.js:1
3                                   pathturbo.js:1
4                                   pathturbo.js:1
5                                   pathturbo.js:1
6                                   pathturbo.js:1
7                                   pathturbo.js:1
8                                   pathturbo.js:1
9                                   pathturbo.js:1
```

While example 2

```
> var counterOne = 10;
  while (counterOne > 0) {
      console.log(counterOne);
      counterOne--;
  }
```

```
10                                  pathturbo.js:1
9                                   pathturbo.js:1
8                                   pathturbo.js:1
7                                   pathturbo.js:1
6                                   pathturbo.js:1
5                                   pathturbo.js:1
4                                   pathturbo.js:1
3                                   pathturbo.js:1
2                                   pathturbo.js:1
1                                   pathturbo.js:1
```

Do-While Loop

```
> var counterTwo = 10;
  do {
      console.log(counterTwo);
      counterTwo--;
  } while (counterTwo > 0);
```

```
10                                                    pathturbo.js:1
9                                                     pathturbo.js:1
8                                                     pathturbo.js:1
7                                                     pathturbo.js:1
6                                                     pathturbo.js:1
5                                                     pathturbo.js:1
4                                                     pathturbo.js:1
3                                                     pathturbo.js:1
2                                                     pathturbo.js:1
1                                                     pathturbo.js:1
```

While vs Do-While Loop

While loop checks condition, then does stuff
Do-while does stuff, then checks condition

```
> var counterOne = 10;
  while (counterOne > 10) {
      console.log(counterOne);
      counterOne--;
  }
```

```
<· undefined
```

```
> var counterTwo = 10;
  do {
      console.log(counterTwo);
      counterTwo--;
  } while (counterTwo > 10);
```

```
10
```

Most of the time >> use For Loop

```
> var toDoList = [
      "clean room",
      "brush teeth",
      "exercise",
      "study js",
      "eat healthy"
  ];
```

forEach method

A simpler way to do for loop. Can also be broken up and use same function with multiple arrays

Instead of doing this:

```
> var toDoListLength = toDoList.length;
  for (var i=0; i < toDoListLength; i++) {
      console.log(toDoList[i], i);
  }
```

Use forEach:

```
> toDoList.forEach(function(toDo, i) {
      console.log(toDo, i);
  })
```

Result:

```
clean room 0
brush teeth 1
exercise 2
study js 3
eat healthy 4
```

Now lets do the same thing, but break up the forEach into a reusable function + forEach…

```
function logToDoList(toDo, i) {
    console.log(toDo, i);
}

toDoList.forEach(logToDoList);
```

```
clean room 0                                    pathturbo.js:1
brush teeth 1                                   pathturbo.js:1
exercise 2                                      pathturbo.js:1
study js 3                                      pathturbo.js:1
eat healthy 4                                   pathturbo.js:1
```

Let's reuse same function for a different array…

Here's the array:

Use forEach method with previous function…

```
> var toDoListOfFool = [
    "messy room!!!",
    "dirty teeth!",
    "sit around!!!!",
    "watch netflix!!",
    "eat junk!!!"
];
```

```
> toDoListOfFool.forEach(logToDoList);
```

```
messy room!!! 0                                 pathturbo.js:1
dirty teeth! 1                                  pathturbo.js:1
sit around!!!! 2                                pathturbo.js:1
watch netflix!! 3                               pathturbo.js:1
eat junk!!! 4                                   pathturbo.js:1
```

We only have to write the function once and can use over and over (extensible). Had we used a For Loop, we would have to re-write the function for each array.

Final version of forEach function that stores the result:

```
function logToDoList(array) {              // takes array as input
      logArray=[];
      array.forEach(function(toDo, i){
            console.log(toDo, i);
            logArray.push(toDo + " " + i);  // adds each to logArray
      })
      return logArray;
}

var resultArray = logToDoList(toDoListOfFool);   // store return array
```

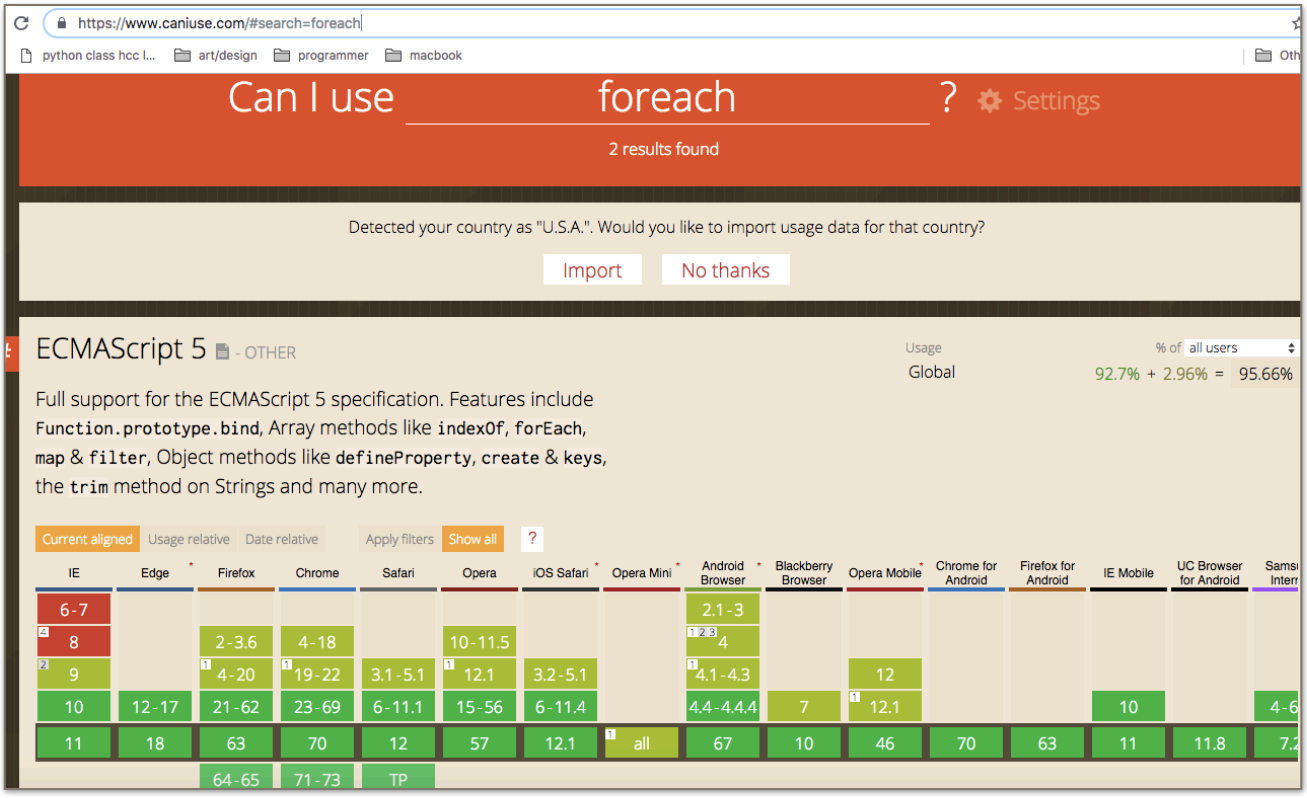
```
> resultArray
<- ▼(5) ["messy room!!! 0", "dirty teeth! 1", "sit around!!!!
    2", "watch netflix!! 3", "eat junk!!! 4"] i
     0: "messy room!!! 0"
     1: "dirty teeth! 1"
     2: "sit around!!!! 2"
     3: "watch netflix!! 3"
     4: "eat junk!!! 4"
     length: 5
   ▶ __proto__: Array(0)
```

A SIMPLER WAY to do this using map( ) instead of forEach( ):

```
var mapResultArray = toDoListOfFool.map(function (toDo, i) {
      return toDo + " " + i;
})
```

```
> mapResultArray
<- ▼(5) ["messy room!!! 0", "dirty teeth! 1", "sit around!!!! 2", "wat
   ch netflix!! 3", "eat junk!!! 4"] i
     0: "messy room!!! 0"
     1: "dirty teeth! 1"
     2: "sit around!!!! 2"
     3: "watch netflix!! 3"
     4: "eat junk!!! 4"
     length: 5
   ▶ __proto__: Array(0)
```

Use www.caniUse.com to see whether forEach is supported across all browsers:



Supported everywhere except partial support in Opera Mini >> good to go!

Advanced JavaScript Arrays

Looping with Array Methods

Example with forEach( )…

```
const numArray = [1, 2, 10, 16];

const doubleArray = [];
const newArray = numArray.forEach(num => {
      doubleArray.push(num*2);// this is a 'side effect' = BAD PRACTICE!
})
// const newArray is unneeded, its always undefined

console.log('forEach result:', doubleArray);
```

```
forEach result: ▶ (4) [2, 4, 20, 32]                    VM3529:10
```

forEach performs actions on the array without storing results. We have to create our own array and push, otherwise don't return anything (undefined). forEach is prone to side effects (it is tempting to use console.log or change an external variable)

map( ) is a method that can replace forEach…

map( )

map( ) loops through an array, calls a function on each element, and stores the result in a new array.

map( ) is better because it:

- expects a return
- Stores results in new array
- Easier to create “pure functions”

Same as previous example, using map( )…

```
var numArray = [1, 2, 10, 16];

// traditional JavaScript
var mapArray = numArray.map(function (num) {
  return num * 2;
});

console.log('map result:',
mapArray);
```

[ES5/6 only]:

```
const numArray = [1, 2, 10, 16];

// ES5/ES6 only
const mapArray = numArray.map (num => num * 2);

console.log('map result:', mapArray);
```

```
map result: ▶ (4) [2, 4, 20, 32]                    VM3529:16
```

filter( )

filter( ) loops through an array while looking at a condition, returns array of only the elements that pass the test (true)

```
var numArray = [1, 2, 10, 16];

var filterArray = numArray.filter(function (num) {
  return num > 5;
});

console.log('filter result:',
filterArray);
```

[ES5/6 only]:

```
const numArray = [1, 2, 10, 16];

const filterArray = numArray.filter(num => num > 5);

console.log('filter result:', filterArray);
```

```
filter result: ▶ (2) [10, 16]                                VM3546:5
```

reduce( )

The **reduce( )** method reduces the array to a single value. It executes a provided function for each value of the array, and the return value of the function is stored in an accumulator.

Sum Example:

```
var numArray = [1, 2, 10, 16];
var reduceSumArray = numArray.reduce(function (accumulator, num) {
  return accumulator + num;
}, 0); // sets starting value of accumulator to zero

console.log('reduce sum result:', reduceSumArray); // 1+2+10+16=29
```

[ES5/6 only]:

```
const reduceSumArray = numArray.reduce((accumulator, num) => {
      return accumulator + num
}, 0) // sets starting value of accumulator to zero

console.log('reduce sum result:', reduceSumArray);
```

```
reduce sum result: 29                                    VM3568:1
```

Subtract from 100 Example:

```
var numArray = [1, 2, 10, 16];
var reduce100Array = numArray.reduce(function (accumulator, num) {
  return accumulator - num;
}, 100); // sets starting value of accumulator to 100

console.log('reduce 100 result:', reduce100Array); // 100-1-2-10-16=71
```

[ES5/6 only]:

```
const reduce100Array = numArray.reduce((accumulator, num) => {
      return accumulator - num
}, 100) // sets starting value of accumulator to 100

console.log('reduce 100 result:', reduce100Array);
```

```
reduce 100 result: 71                                    VM3571:5
```

Exercise: Advanced Arrays

Section 13, Lecture 142

It's time to code some javascript! Get your sublime text ready for this exercise, and use Google Chrome javascript console to test your code. You can find the exercise file and the solution file attached. Good luck!

Also, one of our fellow students Wassim Serhan found this great resource on github for us which we can use to explore array methods. I highly recommend it: https://sdras.github.io/array-explorer/

Resources for this lecture

exercise5-SOLUTIONS (1).js
exercise5.js

// Complete the below questions using this array:

```
const teamArray = [
  {
    username: "john",
    team: "red",
    score: 5,
    items: ["ball", "book", "pen"]
  },
  {
    username: "becky",
    team: "blue",
    score: 10,
    items: ["tape", "backpack", "pen"]
  },
  {
    username: "susy",
    team: "red",
    score: 55,
    items: ["ball", "eraser", "pen"]
  },
  {
    username: "tyson",
    team: "green",
    score: 1,
    items: ["book", "pen"]
  },
];
```

Create an array using forEach that has all the usernames with a "!" to each of the usernames

```
var exclaimArray = [];
teamArray.forEach(function (item) {
  exclaimArray.push(item.username + "!");
    // this is a 'side effect' = BAD PRACTICE!
});
console.log('forEach result:', exclaimArray);


//[ES5/6 only]:
const exclaimArray2 = [];
teamArray.forEach(item => {
  exclaimArray2.push(item.username+"!");
    // this is a 'side effect' = BAD PRACTICE!
})
console.log('forEach result:', exclaimArray2);
```

```
forEach result:                                          VM4610:35
▶ (4) ["john!", "becky!", "susy!", "tyson!"]
```

Create an array using map that has all the usernames with a "? to each of the usernames

```
var mapArray = teamArray.map(function (item) {
  return item.username + "?";
});
console.log('map result:', mapArray);


//[ES5/6 only]:
const mapArray2 = teamArray.map (item => item.username+"?");
console.log('map result:', mapArray2);
```

```
map result: ▶ (4) ["john?", "becky?", "susy?", "tyson?"]   VM4610:54
```

Filter the array to only include users who are on team: red

```
var filterArray = teamArray.filter(function (item) {
  return item.team === "red";
});
console.log('filter result:', filterArray);

//[ES5/6 only]:
const filterArray2 = teamArray.filter(item =>
item.team==="red");
console.log('filter result:', filterArray2);
```

```
filter result:                                                VM4610:61
▼(2) [{…}, {…}]
  ▶0: {username: "john", team: "red", score: 5, items: Array(3)}
  ▶1: {username: "susy", team: "red", score: 55, items: Array(3)}
    length: 2
  ▶__proto__: Array(0)
```

Find out the total score of all users using reduce

```
var reduceSumArray = teamArray.reduce(function (accumulator,
item) {
  return accumulator + item.score;
}, 0); // sets starting value of accumulator to zero

console.log('reduce sum result:', reduceSumArray);

//[ES5/6 only]:
const reduceSumArray2 = teamArray.reduce((accumulator, item) =>
{
  return accumulator + item.score
}, 0) // sets starting value of accumulator to zero

console.log('reduce sum result:', reduceSumArray2);
```

```
reduce sum result: 71                                         VM4610:79
```

BONUS: create a new list with all user information, but add "!" to the end of each items they own.

```
//[ES5/6 only]:
const mapArrayItems = teamArray.map (user => {
    user.items = user.items.map (item => {
        return item + "!";
    })
    return user;
})
console.log('map items result:', mapArrayItems);
```

Make this map function pure:

```
const arrayNum = [1, 2, 4, 5, 8, 9];
const newArray = arrayNum.map((num, i) => {
    console.log(num, i);
    alert(num);
    return num * 2;
})
```

```
// SOLN:
const newArray = arrayNum.map((num, i) => {
  return num * 2;
})
```

What is the value of i?

answer: index of the array

Exercise: Build Facebook 2
Incorporate loops to check username/password against multiple username/passwords

```
> // OUR FACEBOOK BUILD

  var database = [          // array (list)
      {                                 // element 0
          username: "andrei",
          password: "supersecret"
      },
      {                                 // element 0
          username: "sally",
          password: "123"
      },
      {                                 // element 0
          username: "ingrid",
          password: "777"
      }
  ];

  var newsFeed = [         // array (list)
      {                                 // element 0
          username: "Bobby",
          timeline: "So tired from school!"
      },
      {                                 // element 1
          username: "Sally",
          timeline: "Javascript is bad ass"
      }
  ];

  var userNamePrompt = prompt("Enter username");
  var passwordPrompt = prompt("Enter password");
```

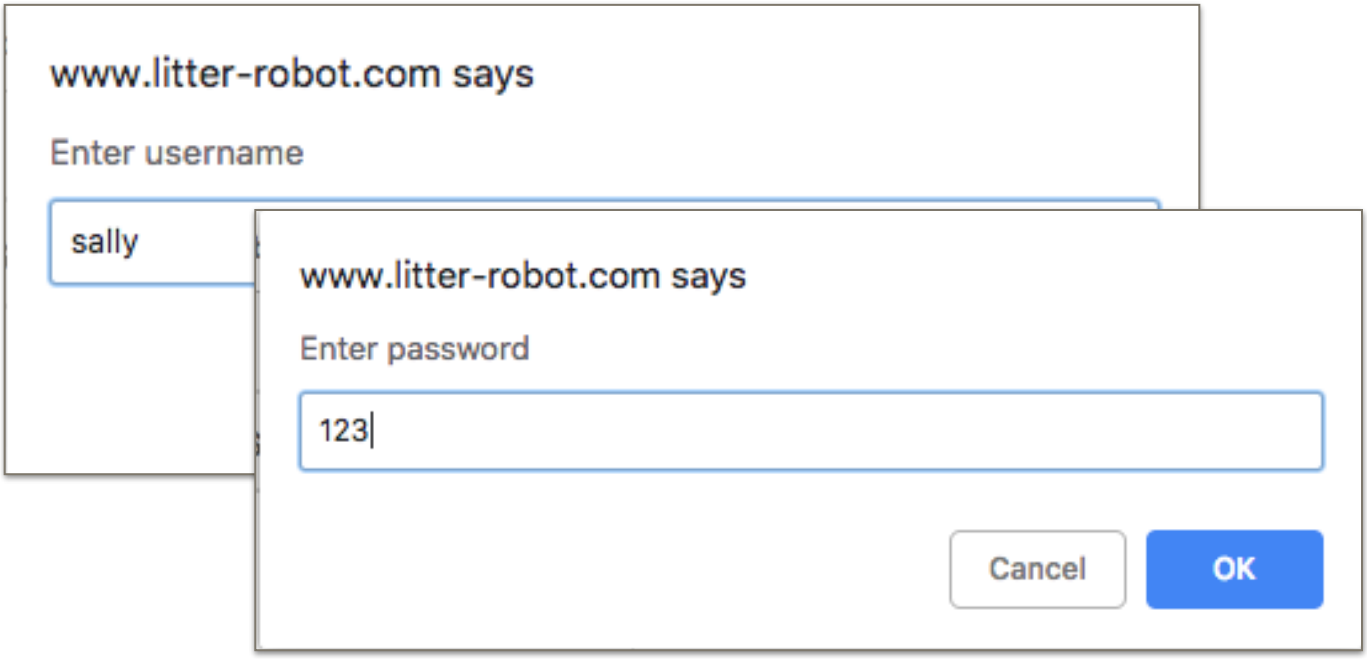
]

```
function isUserValid(user, pass) {
    for (var i=0; i < database.length; i++) {
        if(database[i].username === user &&
            database[i].password === pass) {
            return true; // returns true if un/pw in db
        }
    }
    return false; // returns false if un/pw dont match db
}

function signIn(user, pass) {   // funxn declaration
    // console.log(isUserValid(user,pass)); // logs true
if un/pw correct

    if (isUserValid(user,pass)) {
        console.log(newsFeed); // isUserValid = true
    } else {
        alert("Sorry, wrong UN or PW") // isUserValid =
false
    }

}

signIn(userNamePrompt, passwordPrompt);
```

When we call signIn( ) function in last line of code, the UN and PW entries are passed to the signIn( ) function. The if statement then passes them to the isUserValid( ) function. The isUserValid( ) uses a for loop to compare the parameters to the database, returning true only if they match.

If isUserValid is true, the newsFeed is logged
If isUserValid is false, we get a wrong password alert.

If username and password match a set in database:



Timeline is logged…

If no match to database…