



JavaScript: Higher order functions

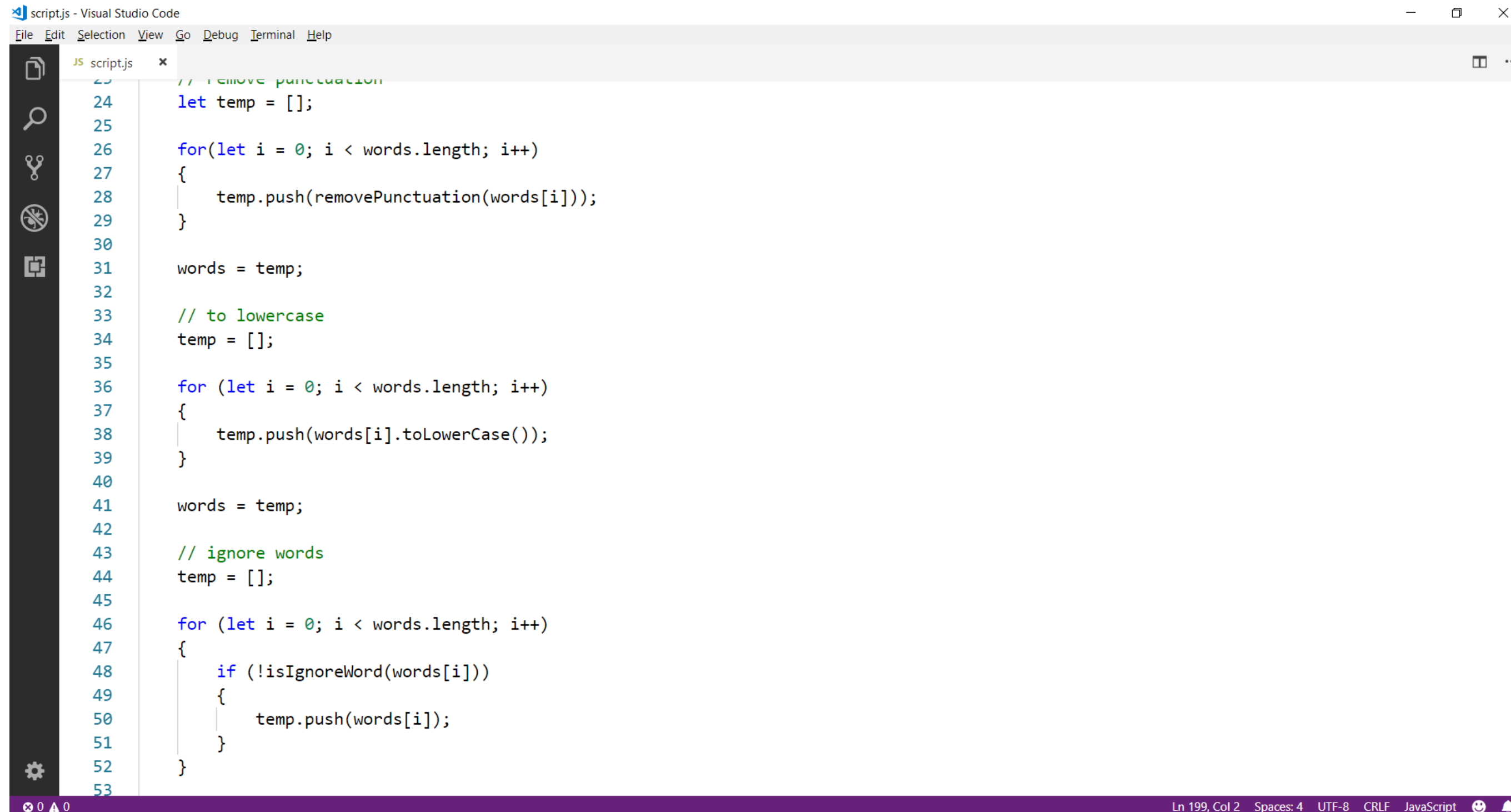
Web, Mobile and Security
Frédéric Vlummens

Let's start with an exercise

- Count the frequency of words in a text.
- Generate a **frequency table**, containing the words and their occurrence count.
- Only words occurring 2+ times should appear in the table.
- It should be possible to ignore common words such as **the**.
- Break up the problem in smaller problems, different steps.
- Don't be afraid to write lots of **small functions** that each do **one thing**.

Demo time

Conclusion: a lot of for loops



The screenshot shows the Visual Studio Code editor with a file named 'script.js'. The code is written in JavaScript and includes several for loops. The editor's interface includes a sidebar on the left with icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The top menu bar shows File, Edit, Selection, View, Go, Debug, Terminal, and Help. The status bar at the bottom indicates 'Ln 199, Col 2', 'Spaces: 4', 'UTF-8', 'CRLF', 'JavaScript', and a smiley face icon.

```
23 // remove punctuation
24 let temp = [];
25
26 for(let i = 0; i < words.length; i++)
27 {
28     temp.push(removePunctuation(words[i]));
29 }
30
31 words = temp;
32
33 // to lowercase
34 temp = [];
35
36 for (let i = 0; i < words.length; i++)
37 {
38     temp.push(words[i].toLowerCase());
39 }
40
41 words = temp;
42
43 // ignore words
44 temp = [];
45
46 for (let i = 0; i < words.length; i++)
47 {
48     if (!isIgnoreWord(words[i]))
49     {
50         temp.push(words[i]);
51     }
52 }
53
```

Mapping an array to another array

- Each element gets converted to another element
- Rule of thumb: length of input and output array are equal

MAP

► (3) [1, 2, 3]

► (3) [2, 3, 4]

>

```
function map(array, func)
{
    let out = [];

    for (let i = 0; i < array.length; i++)
    {
        let elem = array[i];
        out.push(func(elem));
    }

    return out;
}

function plusOne(elem)
{
    return elem + 1;
}

let elems = [ 1, 2, 3 ];

console.log(elems);
elems = map(elems, plusOne);
console.log(elems);
```



Filtering an array

- Elements get included in output array depending on test.
- Length of output array can be less than length of input array.

FILTER

► (6) [1, 2, 3, 4, 5, 6]

► (3) [2, 4, 6]

>

```
function filter(array, func)
{
    let out = [];

    for (let i = 0; i < array.length; i++)
    {
        let elem = array[i];
        if (func(elem))
        {
            out.push(elem);
        }
    }

    return out;
}

function isEven(n)
{
    return n % 2 === 0;
}

let array = [ 1, 2, 3, 4, 5, 6 ];
console.log(array);
array = filter(array, isEven);
console.log(array);
```

Reducing an array

- Array to specific single output value
- Uses an accumulator (=which accumulates values)

REDUCE

```
► (5) [1, 2, 3, 4, 5]
```

```
15
```

```
>
```

```
function reduce(array, func, init)
{
    let out = init;

    for (let i = 0; i < array.length; i++)
    {
        let elem = array[i];
        out = func(out, elem);
    }

    return out;
}
```

```
function sum(accumulator, value)
{
    return accumulator + value;
}
```

```
let array = [1, 2, 3, 4, 5];
console.log(array);
```

```
let result = reduce(array, sum, 0);
console.log(result);
```



map, filter and reduce

- Are built-in methods of Array
- No need to write them ourselves!

```
function timesTwo(elem)
{
    return elem * 2;
}

let input = [1, 2, 3];
console.log(input);
let output = input.map(timesTwo);
console.log(output);
```



```
▶ (3) [1, 2, 3]
▶ (3) [2, 4, 6]
>
```

map, filter and reduce

- Are built-in methods of Array
- No need to write them ourselves!

```
function isEven(elem)
{
    return elem % 2 === 0;
}

let input = [1, 2, 3, 4, 5, 6];
console.log(input);
let output = input.filter(isEven);
console.log(output);
```



```
► (6) [1, 2, 3, 4, 5, 6]
► (3) [2, 4, 6]
>
```


map, filter and reduce

- Are built-in methods of Array
- No need to write them ourselves!

```
function add(accumulator, elem)
{
    return accumulator + elem;
}

let input = [1, 2, 3, 4, 5 ];
console.log(input);
let output = input.reduce(add, 0);
console.log(output);
```



```
► (5) [1, 2, 3, 4, 5]
15
>
```

Other useful array methods: forEach

- Applies a function to each element of the array

```
let array = [1, 2, 3, 4, 5];  
  
function output(elem)  
{  
    console.log(elem);  
}  
  
array.forEach(output);
```



1
2
3
4
5
>

Other useful array methods: sort

- Sorts an array
- Provide a comparison function

```
function compare(a, b)
{
    return a - b;
}

let input = [ 3, 7, 2, 5, 1 ];
console.log(input);

let output = input.sort(compare);
console.log(output);
```



```
► (5) [3, 7, 2, 5, 1]
► (5) [1, 2, 3, 5, 7]
>
```

Other useful array methods: find and findIndex

- **find**: returns value of first element in array that satisfies provided testing function. Returns undefined if no element found.
- **findIndex**: returns index of first element in array that satisfies provided testing function. Returns -1 if no element found.

```
function isEven(elem)
{
    return elem % 2 === 0;
}

let array = [1, 2, 3 ];

console.log( array.find(isEven) );

console.log( array.findIndex(isEven) );
```



```
2
1
>
```

Arrow functions

- Shorter syntax than function declaration
- Ideal for use with higher order functions
- Example:

```
let input = [ 1, 2, 3, 4, 5 ];  
console.log(input);  
  
let output = input.map(e => e * 2);  
console.log(output);
```



```
▶ (5) [1, 2, 3, 4, 5]  
▶ (5) [2, 4, 6, 8, 10]  
>
```

Arrow functions

- Can have multiple arguments where required
- Example:

```
let input = [ 3, 7, 2, 5, 1 ];  
console.log(input);  
  
let output = input.sort( (a, b) => a - b );  
console.log(output);
```



```
▶ (5) [3, 7, 2, 5, 1]  
▶ (5) [1, 2, 3, 5, 7]  
>
```

Chaining

- Since map and filter return an array, you can call array functions on the result of these function calls and create a **chain**.

```
let array = [1, 2, 3, 4, 5];  
console.log(array);  
  
let output = array.map(elem => elem * 3)  
                  .filter(elem => elem % 2 === 0);  
  
console.log(output);
```



```
► (5) [1, 2, 3, 4, 5]  
► (2) [6, 12]  
>
```

Let's practice

- Using all knowledge we've gained, let's refactor our code to use map, filter, reduce and other Array methods at our disposal.
- We shall keep our original JavaScript source, so we can compare later on between both versions!

Demo time

Questions?

