MIND HUB.

Higher Order Functions

Index

Definition

Functions that operate on other functions, either by taking them as arguments or by returning them.

It is higher order because instead of strings, numbers or Boolean values, it operates on functions.

This is a concept born out of the functional programming paradigm.

They serve to hide the detail, that is, they provide a higher level of abstraction.

Advantages

Simple and elegant code

Higher-order functions allow you to write simple, clean code. It allows you to write smaller functions, which do only one thing. This type of composition results in simple, readable code.

Less errors

With simple and elegant code you will end up with fewer errors.

Easy to test and debug

With functions that only do one thing, you end up with code that is easy to test. Testable code that generates fewer errors. Therefore, debugging these simple functional units is also easy.

Integrated to JS

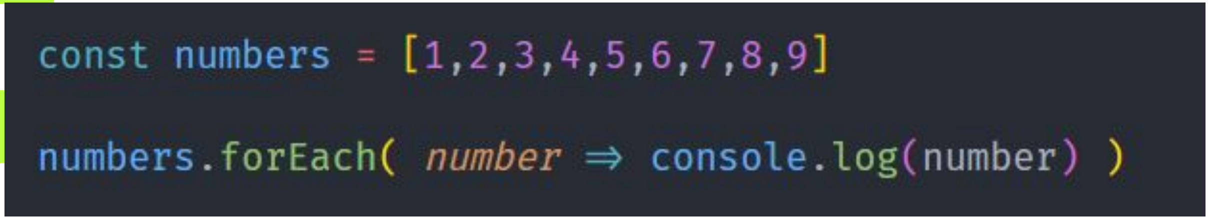
JavaScript comes with some built-in higher-order functions. You may already be using them, without realizing that they are higher-order functions. Let's look at some of them to understand how they work.

- .map()
- .forEach()
- .filter()
- .reduce()
- .find()
- .findIndex()
- .some()
- .sort()

There are many more but for now let's concentrate on these and analyze their advantages

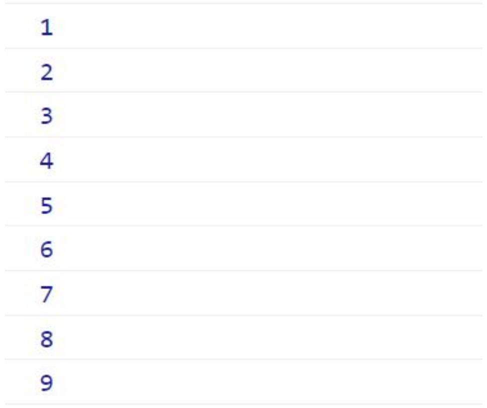
.forEach(fn)

Traverses the array and executes the callback function for each element in the array, always returns undefined



```javascript
const numbers = [1,2,3,4,5,6,7,8,9]

numbers.forEach( number => console.log(number) )
```

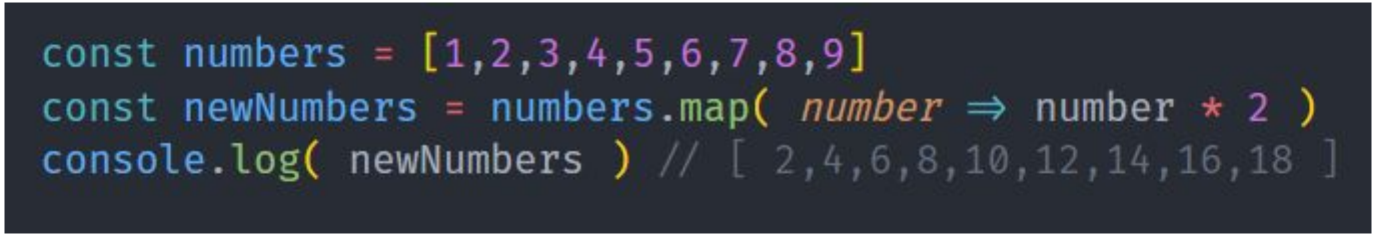
Result



```
1
2
3
4
5
6
7
8
9
```

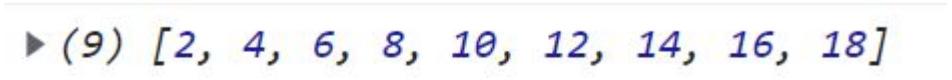
.map(fn)

Iterates the array and executes the callback function for each element in the array, returns a new array which can be stored in a variable or used as argument on another function



```
const numbers = [1,2,3,4,5,6,7,8,9]
const newNumbers = numbers.map( number ⇒ number * 2 )
console.log( newNumbers ) // [ 2,4,6,8,10,12,14,16,18 ]
```

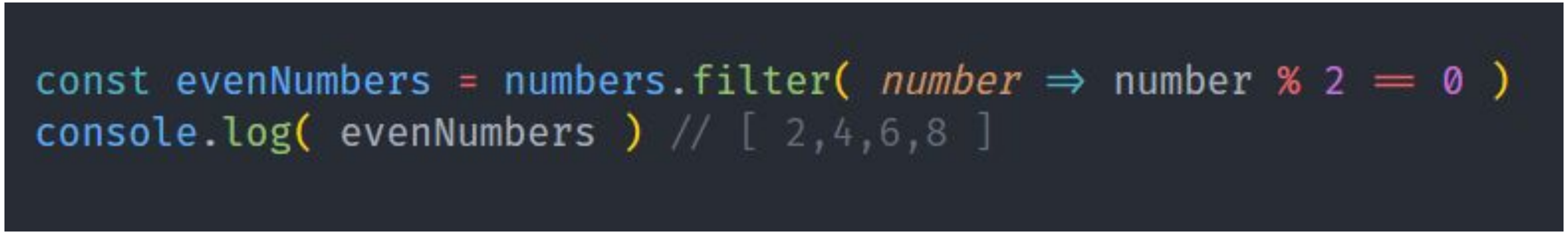
Result



```
▶ (9) [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

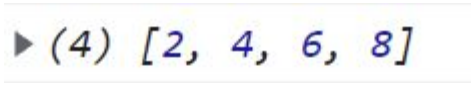
.filter()

Traverses an array and executes the argument function for each element of the array. If it returns a truthy value that element is stored in a new array, if it returns a falsy the element isn't stored. When the traversing ends that array can be stored in a variable or used as argument in another function.



```
const evenNumbers = numbers.filter( number => number % 2 === 0 )
console.log( evenNumbers ) // [ 2,4,6,8 ]
```

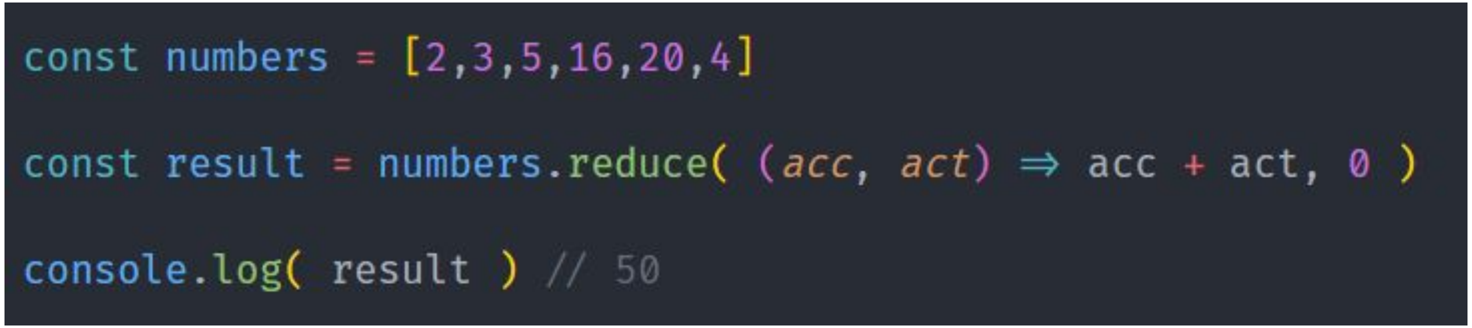
Result



```
▶ (4) [2, 4, 6, 8]
```

.reduce(fn, initialValue)

Traverses an array and executes the argument function for each element in the array. It allows us to work with an accumulator which will contain the previous' iteration return, when the traversing ends the accumulator is returned.

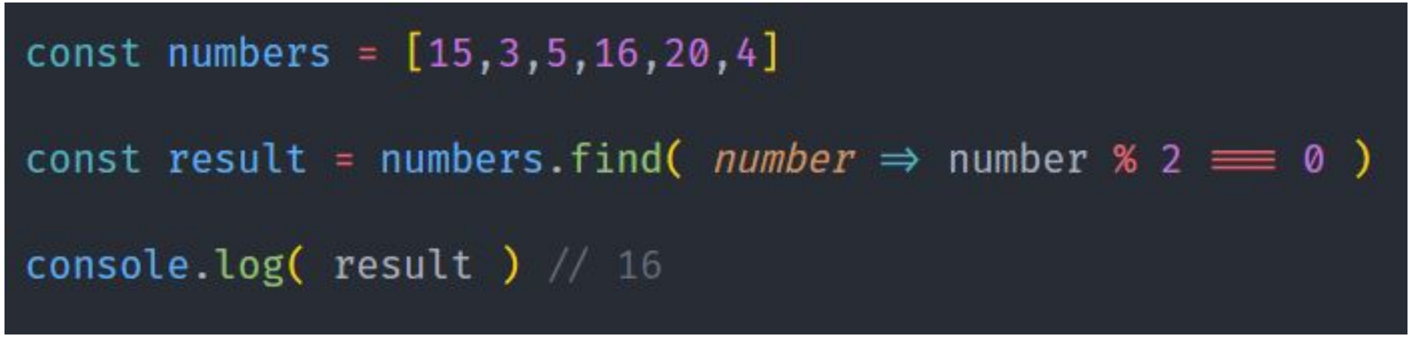


```
const numbers = [2,3,5,16,20,4]

const result = numbers.reduce( (acc, act) ⇒ acc + act, 0 )

console.log( result ) // 50
```

An initial value can be sent for the accumulator as a second argument, without it the initial value will be the first element of the array.

.find(fn)

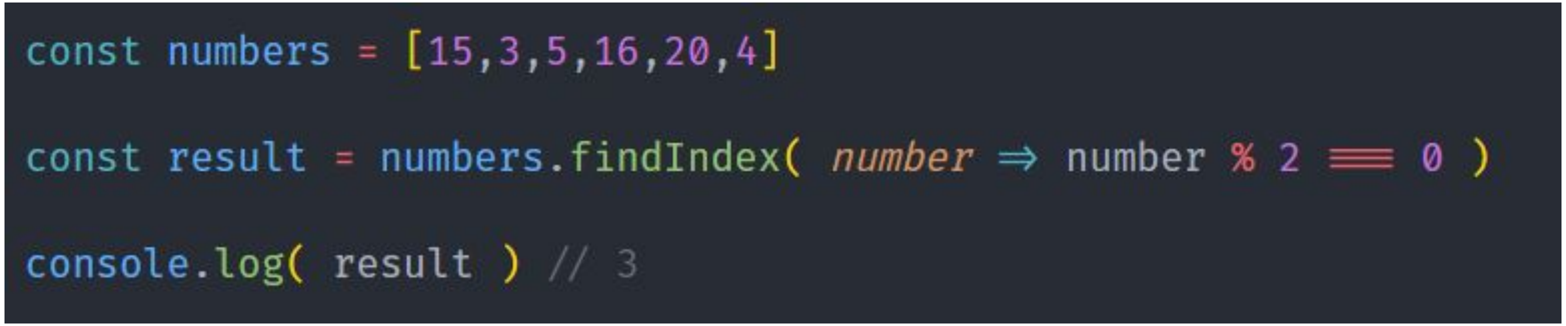
Traverses an array and executes the argument function for each element in the array. If the function returns a truthy the iteration stops and the actual element is returned.



```
const numbers = [15,3,5,16,20,4]

const result = numbers.find( number => number % 2 === 0 )

console.log( result ) // 16
```

.findIndex(fn)

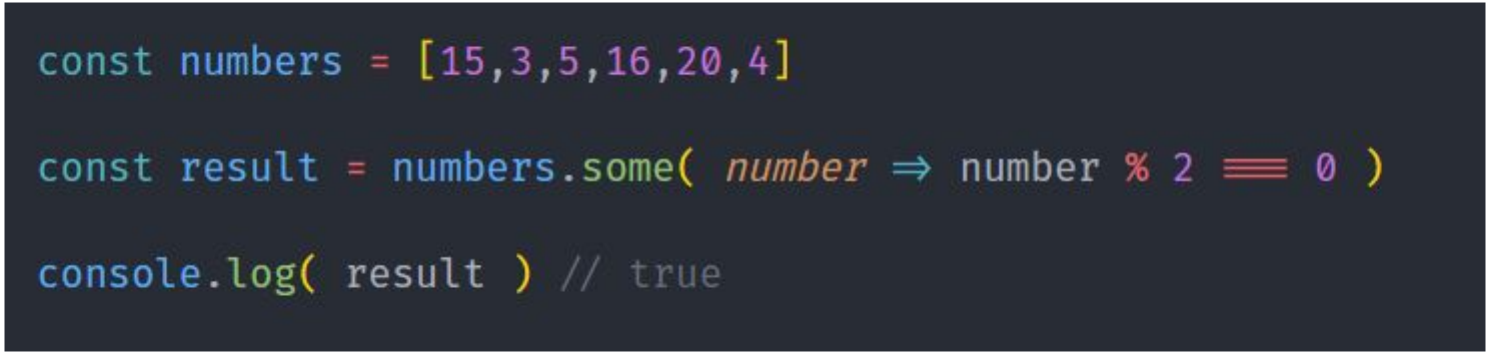
Traverses an array and executes the argument function for each element in the array. If the function returns a truthy the iteration stops and the actual element’s index is returned.



```
const numbers = [15,3,5,16,20,4]

const result = numbers.findIndex( number => number % 2 === 0 )

console.log( result ) // 3
```

.some(fn)

Traverses an array and executes the argument function for each element in the array. If the function returns a truthy the iteration stops and returns a true, otherwise returns false.

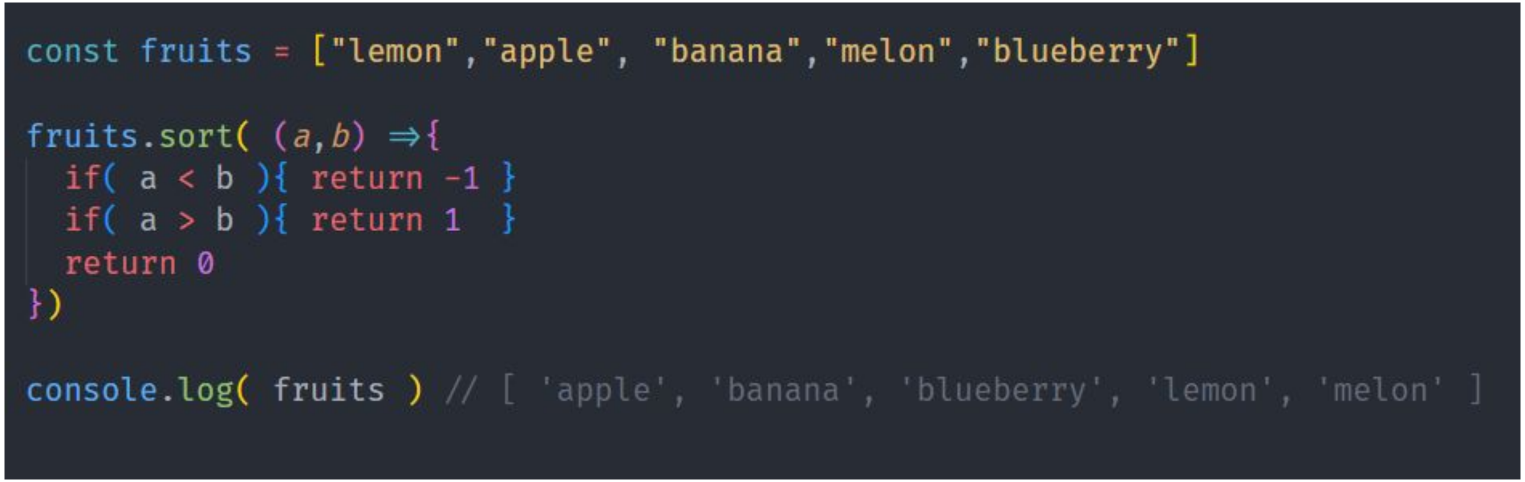


```
const numbers = [15,3,5,16,20,4]

const result = numbers.some( number => number % 2 === 0 )

console.log( result ) // true
```

.sort(fn)

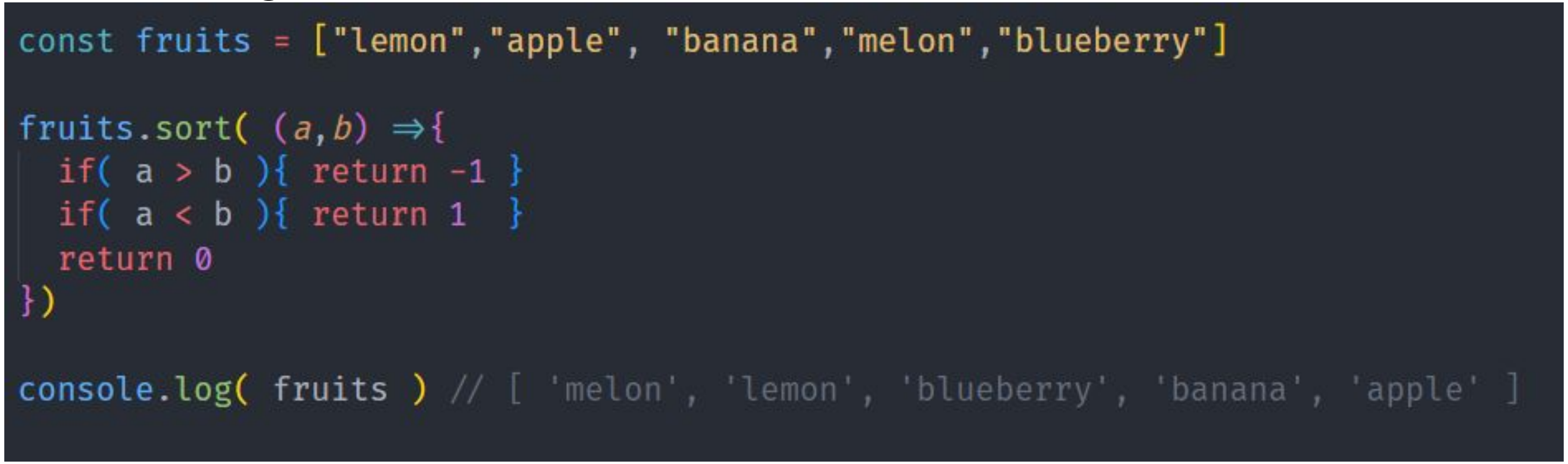
Sort requires the use of a sort function to perform ascending and descending sorting or to sort letters or numbers. It modifies the original array

ascending:

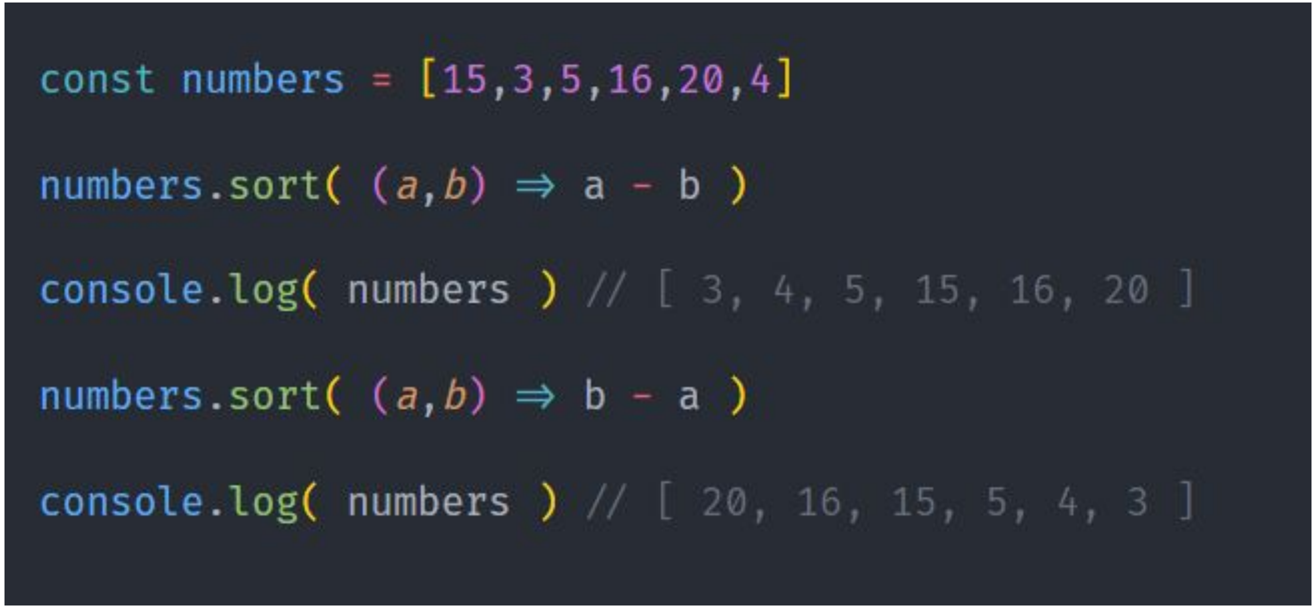


```
const fruits = ["lemon","apple", "banana","melon","blueberry"]

fruits.sort( (a,b) =>{
  if( a < b ){ return -1 }
  if( a > b ){ return 1  }
  return 0
})

console.log( fruits ) // [ 'apple', 'banana', 'blueberry', 'lemon', 'melon' ]
```

descending:



```
const fruits = ["lemon","apple", "banana","melon","blueberry"]

fruits.sort( (a,b) =>{
  if( a > b ){ return -1 }
  if( a < b ){ return 1  }
  return 0
})

console.log( fruits ) // [ 'melon', 'lemon', 'blueberry', 'banana', 'apple' ]
```



```
const numbers = [15,3,5,16,20,4]

numbers.sort( (a,b) ⇒ a - b )

console.log( numbers ) // [ 3, 4, 5, 15, 16, 20 ]

numbers.sort( (a,b) ⇒ b - a )

console.log( numbers ) // [ 20, 16, 15, 5, 4, 3 ]
```

MIND
HUB.