**Iterators**

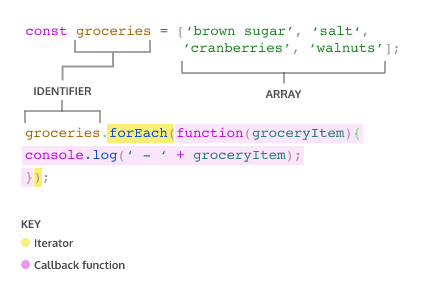
There are built-in JavaScript array methods that help us iterate without the use of for loops, called iterators. Iterators are methods called on arrays to manipulate elements and return values.

**Iterators takes function as arguments (asynchronous callback functions)**

We can also use the => syntax and function expression

We can also define the function before hand to be used as the callback function.

1. **.forEach() method**



+) .forEach() takes an argument of callback function.

+) .forEach() loops through the array and executes the callback function for each element. During **each execution, the current element is passed as an argument to the callback function** *(groceryItem)*

+) **Return value for .forEach() will always be undefined**.

1. **.map() method**

When map is called on an array, it also takes argument of a callback function, but **returns a new array that contains the same number of elements.** If some need to filter out, use .filter() instead.

NOTE: the results returned from each iteration will be automatically put into an array. Therefore, we need to **assign <array>.map() to some variables**.

const animals = ['Hen', 'elephant', 'llama', 'leopard', 'ostrich', 'Whale', 'octopus', 'rabbit', 'lion', 'dog'];

const secretMessage = animals.map(current => {

  return current[0];

)}

console.log(secretMessage.join(''));

//HelloWorld

1. **.filter() method**

Like .map(), .filter() returns an array. But .filter() **returns an array after filtering out some elements**.

The **callback function** for the .filter(0 method **should return true or false** depending on the element passed to it. The elements that cause the callback function to return true are added to the array.

const words = ['chair', 'music', 'pillow', 'brick', 'pen', 'door'];   
  
const shortWords = words.filter(word => {  
  return word.length < 6;  
});

1. **.findIndex() method**

Calling .findIndex() on an array will **return the index** of the first element that **evaluates to true** in the callback function. If no element satisfies, return -1

const jumbledNums = [123, 25, 78, 5, 9];   
  
const lessThanTen = jumbledNums.findIndex(num => {  
  return num < 10;  
});

1. **.reduce() method**

The .reduce() method **returns a single value** after iterating through the elements of an array, thereby *reducing* the array.

* The callback funtion has 2 parameters, accumulator and currentValue. The value of **accumulator starts off as the value of the first element in the array and the currentValue starts as the second element**.
* As .reduce() iterates through the array, the return value of the callback function becomes the accumulator value for the next iteration, currentValue takes on the value of the current element in the looping process (starts at second element in array)
* The method can also take an optional **second parameter** to set an **initial value for accumulator**

This function can be used for summing or multiplying elements in an array.

Text

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

1. **.some() and .every() method**

These two methods **returns a boolean value**.

+) .some() will evaluate if SOME elements **satisfy a condition**

+) .every() will evaluate if EVERY elements **satisfy a condition**

const jumbledNums = [123, 25, 78, 5, 9];   
  
const someLessThanTen = jumbledNums.some(num => num < 10);

//true