**map**

var arr = [1, 2, 3, 4, 5];

var arr2 = arr.map(value => value \* 2);

**reduce**

* Same as map but accept function with previous and current element as parameter

var number = [1, 2, 3, 4, 5, 6, 7, 8, 9];

var total = number.reduce(function(previous, current) {

return previous + current;

});

var array\_of\_arrays = [[1, 2], [3, 4], [5, 6]];

var concatenated = array\_of\_arrays.reduce(function(previous, current) {

return previous.concat(current);

});

var numbers = [65, 44, 12, 4];

function getSum(total, num) {

return total + num;

}

function myFunction(item) {

document.getElementById("demo").innerHTML = numbers.reduce(getSum);

}

**Filters**

--------

var tasks =[

{

'name' : 'Task 1',

'duration' : 120

},

{

'name' : 'Task 2',

'duration' : 100

},

{

'name' : 'Task 3',

'duration' : 99

},

];

var taskst = tasks.filter(x => x.duration >100 || x.duration ==100);

**Benefits of React**

* Simpler :
  + Component base architecture with pure Javascript
  + Declarative style not Imperative
  + Developer friendly DOM abstraction
* Fast UIs
  + Virtual DOM - updated only changed element (JQuery – Manual, Angular – All automated all the time)
* Less code to write – Component reuse

Component based Architecture

* Separation of concern
* Loose coupling
* Code reuse

React library has two packages (Version 0.14)

1. React Core (**react** package on npm)
2. ReactDOM (**react-dom** package on npm)

Because of this split rendering syntax got changes

* Before 0.14 (in 0.13)
  + React.render()
* After 0.14
  + ReactDOM.render()

React is almost used with JSX – a tiny language that let developers write React UIs more eloquently. We can transpile JSX into regular Javascript by using **Babel** or a similar tool.