**Async-you**

<https://www.engineyard.com/blog/taming-asynchronous-javascript-with-async>

<https://caolan.github.io/async/docs.html>

1. **Waterfall**

One thing is passed in >> a sequence of functions process the thing, passing the result of processing from one to the next >> the last function output is returned as the result of sequential processing.

<https://www.npmjs.com/package/async-waterfall>

1. **Series**

One thing is passed in >> the thing is given to a number (array or object) of functions that process the thing independently >> the array of independent outputs is returned as the result of independent processing.

<https://www.npmjs.com/package/async-series>

1. **Each**

A number of things is passed in as an array >> the array items are processed by one function making sure that there is no error happens anywhere on the way >> the error is returned in case it happens.

<https://www.npmjs.com/package/async-each>

1. **Map**

A number of things is passed in as an array >> the array items are processed by one function that sticks the outputs of the each item processing into the array of the outputs >> the array of the outputs is returned.

<https://www.npmjs.com/package/async-map>

1. **Times**

The alternative of the for() loop in async. First argument is the number of iterations. The second is the function to execute. The outputs of the executions are put into the array. The array is returned.

<https://caolan.github.io/async/docs.html#times>

1. **Reduce**

Array of items is passed in; start value is passed in as memo>>the function called for each item of the array, passing back the accumulated memo>>the last function just brings the result or an error.

<https://www.npmjs.com/package/async-reduce>

<https://caolan.github.io/async/docs.html#reduce>

1. **Whilst**

Three functions:

1. test() – test (test logic is in the function) each time before work function is called; if true returned – work is executed, if false returned – jump to the result.
2. work() – all the work is done here (the work logic in the function); the callback returns the result of the work and either applies it again in the next work() or passes it to the result();
3. result() – returns error or the result of the whole whilst.

<https://github.com/async-js/async#whilst>