Chapter 83: How to make iterator usable inside async callback function

When using async callback we need to consider scope. **Especially** if inside a loop. This simple article shows what not to do and a simple working example.

Section 83.1: Erroneous code, can you spot why this usage of key will lead to bugs?

```
var pipeline = {};
// (...) adding things in pipeline

for(var key in pipeline) {
  fs.stat(pipeline[key].path, function(err, stats) {
    if (err) {
      // clear that one
      delete pipeline[key];
      return;
    }
    // (...)
    pipeline[key].count++;
  });
}
```

The problem is that there is only one instance of **var key**. All callbacks will share the same key instance. At the time the callback will fire, the key will most likely have been incremented and not pointing to the element we are receiving the stats for.

Section 83.2: Correct Writing

```
var pipeline = {};
// (...) adding things in pipeline

var processOneFile = function(key) {
  fs.stat(pipeline[key].path, function(err, stats) {
    if (err) {
      // clear that one
      delete pipeline[key];
      return;
    }
    // (...)
    pipeline[key].count++;
  });
};

// verify it is not growing
for(var key in pipeline) {
  processOneFileInPipeline(key);
}
```

By creating a new function, we are scoping **key** inside a function so all callback have their own key instance.