**Source**: learnyounode from nodeschool.io

* To make a Node.js program
  + bla.js file - execute like node bla.js
* To verify run
  + learnyounode verify myprogram.js
* To just run with test environment
  + learnyounode run myprogram.js
* To access command-line arguments
  + process object (global) - has argv (process.argv)
* Convert to number
  + prefix with + or Number()
* To include a module
  + var fs = require('fs')
* To install a node package
  + Npm install whatever
* fs info
  + - All synchronous (blocking) methods end with "Sync". use fs.readFileSync('/path/to/file'). it returns a Buffer object with the complete contents. Second argument can be encoding (will return string)
  + - To get contents do var str = buf.toString()
  + - use split to count lines
* To create a module
  + **module.exports** = function (args) {/\*\*/} – module.exports is what is actually returned when you require a module
  + (use require('./moduleName'))
  + if you export a single function you can call the module directly (moduleName())
  + Idiomatic to do early returns
    - function bar (callback) {
    - foo(function (err, data) {
    - if (err)
    - return callback(err) // early return
    - // ... no error, continue doing cool things with `data`
    - // all went well, call callback with `null` for the error argument
    - callback(null, data)
    - })
    - }
* HTTP Get Request
  + http = require('http');
  + http.get(url, callback);
    - function callback (response){}
      * + response is a **Stream** object (emits events, the three events that are of most interest are: "data", "error" and "end".)
        + stream.on("data", function (data) { /\* ... \*/ })
        + the http one also has a setEncoding() method.
        + Use a third-party package to abstract the difficulties involved in collecting an entire stream of data. Two different packages provide a useful API for solving this problem (there are likely more!): **bl** (Buffer List) and **concat-stream**; take your pick!
        + Counting callbacks is one of the fundamental ways of managing async in Node. Rather than doing it yourself, you may find it more convenient to rely on a third-party library such as [async](http://npm.im/async) or [after](http://npm.im/after). But for this exercise, try and do it without any external helper library.
* TCP server
  + var net = require('net')
  + var server = net.createServer(function (socket) {
  + // socket handling logic
  + })
  + server.listen(8000)
  + Unlike most callbacks in Node, the callback used by createServer() is called more than once. Every connection received by your server triggers another call to the callback.
  + The socket object contains a lot of meta-data regarding the connection, but it is also a Node duplex Stream, in that it can be both read from, and written to. For this exercise we only need to write data and then close the socket.
  + Use socket.write(data) to write data to the socket and socket.end() to close the socket. Alternatively, the .end() method also takes a data object so you can simplify to just: socket.end(data).
* HTTP Server
  + Request and response are also streams
  + fs.createReadStream() can create a stream
  + use **src.pipe(dst)** for streams
  + Code
    - var http = require('http')
    - var server = http.createServer(function (req, res) {
    - // request handling logic...
    - })
    - server.listen(8000)
  + Transforming data
    - Package through2-map
      * var map = require('through2-map')
      * inStream.pipe(map(function (chunk) {
      * return chunk.toString().split('').reverse().join('')
      * })).pipe(outStream)
  + Parse url
    - var urlParser = require('url');
    - var server = http.createServer(function (req, res) {
    - if(req.method == "GET"){
    - var urlParts = urlParser.parse(req.url,true);
    - var path = urlParts.pathname;
    - var query = urlParts.query;