Wifi mit Lua

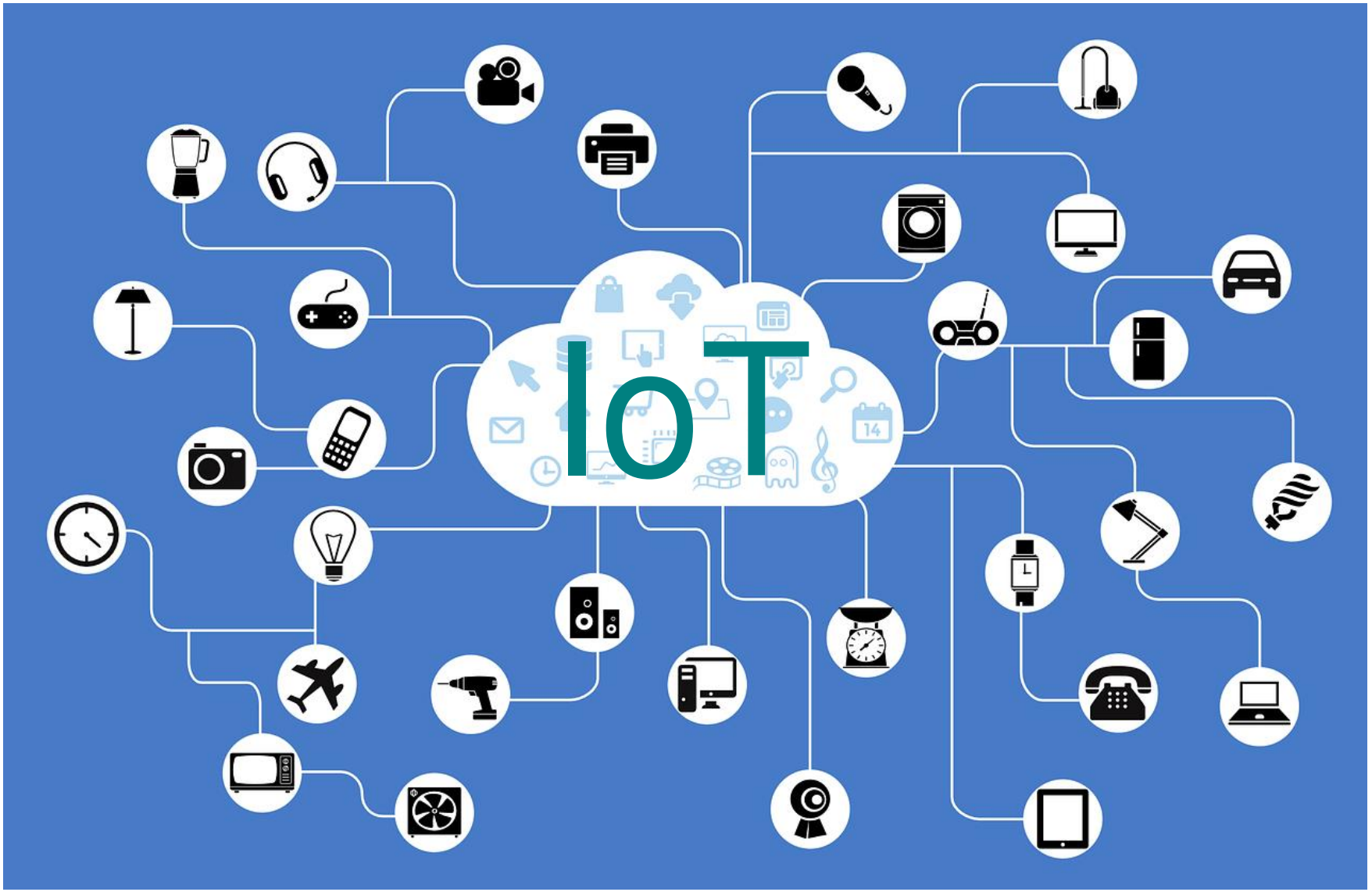
NodeMCU für ESP8266-Module

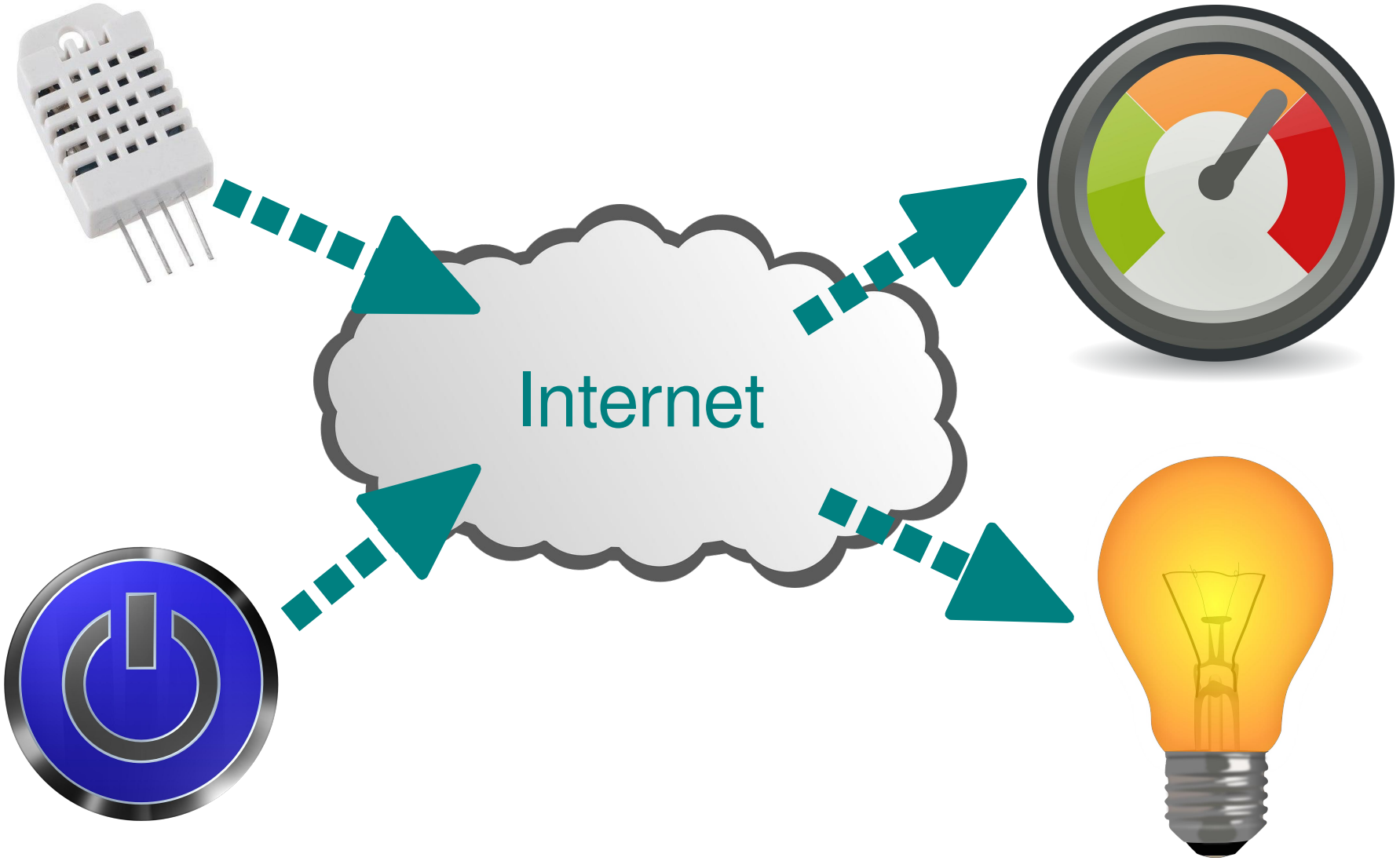
Uwe Berger

bergeruw@gmx.net

Uwe Berger

- Beruf: Softwareentwickler
- Freizeit: u.a. mit Hard- und Software rumspielen
- Linux seit ca. 1995
- BraLUG e.V.
- bergeruw@gmx.net


IoT

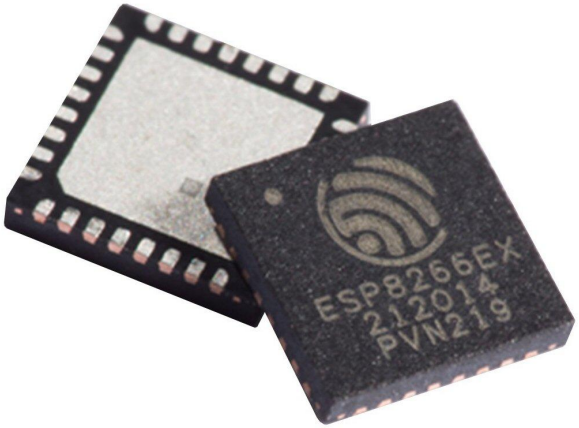

Internet

```
-- WLAN konfigurieren/einschalten
wifi.setmode(wifi.STATION)
wifi.sta.config("MEIN_WLAN", "meinpasswort")
wifi.sta.connect()


-- DHT11 auslesen
function read_dht11()
    local dht11_pin = 4
    status, temp, humi, temp_dec, humi_dec = dht.read(dht11_pin)
end

-- DHT11 zyklisch jede Sekunde auslesen
tmr.alarm(2, 1000, 1, function() read_dht11() end)


-- HTTP-Server
srv=net.createServer(net.TCP)
srv:listen(80, function(conn)
                conn:on("receive", function(conn,request)
                                   local buf=""..temp.."&deg;C|"..humi.."%"
                                   conn:send(buf)
                                   conn:close()
                                   end)
                end)
```

Der Weg dahin:

- Ein Stück Hardware → ESP8266
- Eine Toolchain:
  - Lua-Firmware → NodeMCU
  - Flash- und Upload-Tools
- Etwas Verständnis für Lua allgemein und „NodeMCU-Lua“ im Speziellen
- Ein paar allgemeine Beispiele

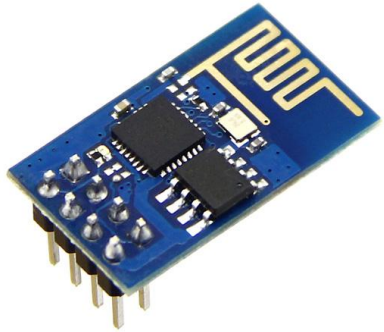
ESP8266-Chip



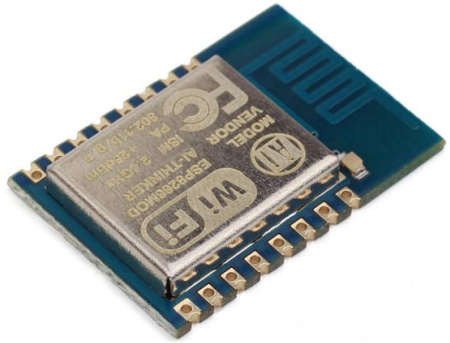
- Hersteller: Espressif (China)
- SoC (System of a Chip) mit:
  - CPU: 32-Bit RISC (Xtensa LX106), 80...160MHz
  - Speicher:
    → interner RAM: 64kByte + 94kByte
    → extern: SPI-Schnittstelle für Flash-Speicher (bis 128MBit)
  - WLAN etc.:
    → 802.11 b/g/n (Tx: +20...+14dBm; Rx:-91...-72dbm)
    → WPA/WPA2; WEP/TKIP/AES
    → Wifi-Mode: Station, SoftAP, Station+SoftAP
    → Built-in: TCP/IP protocol stack (IPv4)
  - UART, I2C, PWM, SPI, GPIOs, ADC
  - 3,3V, 0,01mA...250mA (sleep...send)

ESP8266-Module

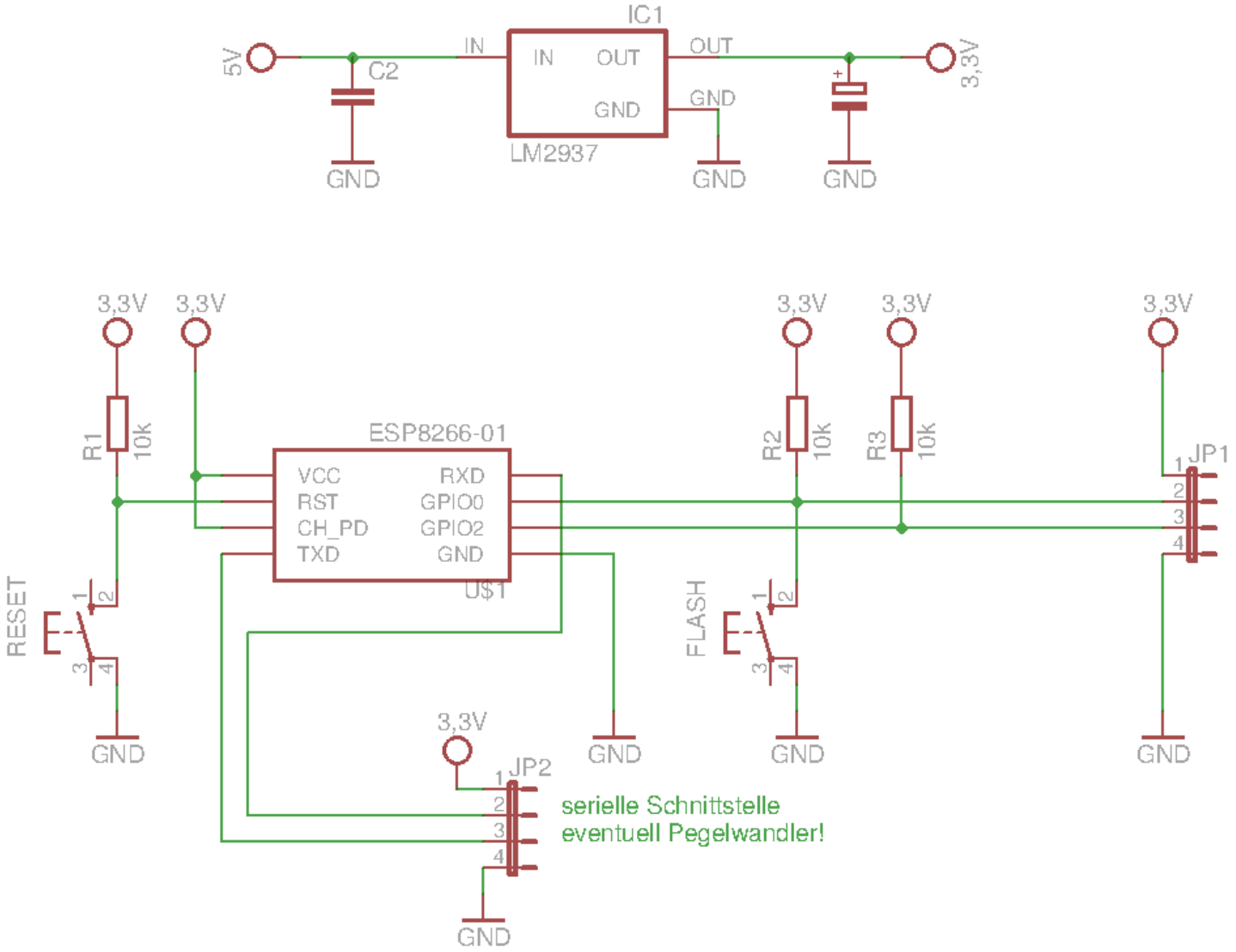
- Diverse Ausführungsformen, Unterschied u.a.:
  → Anzahl herausgeführter Schnittstellen
  → Größe des externen Flash-Speichers
  → Ausführung Antenne
- Beispiele:
  - ESP-01
    → UART, RESET, 2 GPOIs
    → 512 kByte Flash



  - ESP-12E
    → UART, RESET, ADC, SPI, 9 GPIOs
    → 4 MByte Flash



Das „Drumherum“...



Firmware NodeMCU

- Firmware auf Basis des eLua-Projektes und dem ESP8266-SDK (Espressif)
- https://github.com/nodemcu/nodemcu-firmware
- Lizenz: MIT
- Ermöglicht die Programmierung von ESP8266-Modulen mit der Interpretersprache Lua („on-chip“)
- Bestandteile:
  - Lua-Interpreter-Shell (erreichbar über UART)
  - Filesystem im Flash-Speicher, in dem die Lua-Scripte auf dem Modul liegen

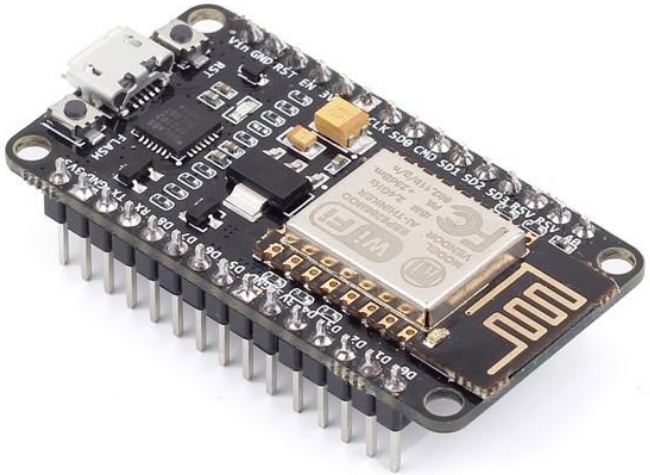
Firmware NodeMCU flashen

- NodeMCU-Firmware konfigurieren/übersetzen, z.B.:
  → https://github.com/nodemcu/nodemcu-firmware
  → https://hub.docker.com/r/marcelstoer/nodemcu-build/
  → https://nodemcu-build.com/
- NodeMCU auf ESP-Modul flashen:
  - Über Hardware-UART des ESP8266-Modul
  - Mit Hilfe spezieller Flash-Tools, z.B. `esptool.py`
    → https://github.com/espressif/esptool

```
# Beispiel fuer ESP-01...
esptool.py --port /dev/ttyUSB0 write_flash 0x00000 <nodemcu-firmware>.bin
```

Alternative zum selber flashen...

- Diverse Anbieter verkaufen sogenannte NodeMCU-Boards
  - Firmware NodeMCU bereits aufgespielt
  - Seriell-zu-USB-Wandler und Stromversorgung onboard
- Nachteil: Firmware nicht immer aktuell
  → irgendwann muss man also doch selbst flashen...



Toolchain

- Toolchain zum Lua-Programmieren mit NodeMCU:
  - Editor (eventuell mit Lua-Syntax-Highlighting)
  - Upload-Tool (für die Lua-Scripte)
  - (Eventuell) Terminal für serielle Schnittstelle
- Upload-Tool:
  - Eine gute Übersicht:
    → http://frightanic.com/iot/tools-ides-nodemcu/
  - Ein gutes und vollständiges Kommandozeilentool:
    → nodemcu-uploader
    → https://github.com/kmpm/nodemcu-uploader

Live-Demo?

Lua-KnowHow

- https://www.lua.org/
- „Programmieren in Lua“; Ierusalimschy, Roberto; Open Source Press, September 2006; ISBN 3937514228
- Vortrag „Die Scriptsprache Lua“ (Uwe Berger) ;-)
- https://nodemcu.readthedocs.io/en/master/

Wie „tickt“ NodeMCU

- Nach Start ESP-Modul:
  - Suche nach Lua-Script `init.lua` im Filesystem
    → vorhanden: Abarbeitung von `init.lua`
    → nicht vorhanden: Interpreter-Shell (via UART)
- Mit dem Lua-Befehl `dofile("script.lua")` können Scripte manuell (bzw. aus `init.lua`) gestartet werden...
- NodeMCU arbeitet ereignisgesteuert und asynchron
  → Erklärung gleich bei den Beispielen

Eine typische init.lua

```
-- Parameter WLAN/IP
ssid, pwd, ip, nm, gw = "x", "y", "10.1.1.43", "255.255.255.0", "10.1.1.1"

-- WLAN konfigurieren und verbinden
print("Connecting to wifi...")
wifi.setmode(wifi.STATION)
wifi.sta.config(ssid, pwd)
wifi.sta.setip({ip = ip, netmask = nm, gateway = gw})
wifi.sta.connect()

-- Warten bis mit WLAN verbunden
tmr.alarm(0, 1000, 1,  function()
                           print(".")
                           local ip = wifi.sta.getip()
                           if ((ip ~= nil) and (ip ~= "0.0.0.0")) then
                               print(ip)
                               tmr.stop(0)
                               -- naechstes Lua-Script starten
                               dofile("weather_clock.lc")
                           end
                       end)
```

Ein kleiner Webserver

```
-- ...init.lua...

-- DHT11 auslesen
function read_dht11()
    local dht11_pin = 4
    status, temp, humi, temp_dec, humi_dec = dht.read(dht11_pin)
end

-- DHT11 zyklisch jede Sekunde auslesen
tmr.alarm(2, 1000, 1, function() read_dht11() end)


-- HTTP-Server
srv=net.createServer(net.TCP)
srv:listen(80, function(conn)
               conn:on("receive", function(conn,request)
                                  local buf=""..temp.."&deg;C|"..humi.."%"
                                  conn:send(buf)
                                  conn:close()
                                  end)
               end)
```

Datum/Uhrzeit aus dem Internet holen

```
-- Zeit holen
function read_ntp()
    net.dns.resolve("de.pool.ntp.org",
        function(sk, ip)
            if (ip == nil) then print("DNS failed!") else
                sntp.sync(ip,
                    function(sec, usec, server)
                        print('sync', sec, usec, server)
                        rtctime.set(sec, usec)
                    end,
                    function() print("NTP sync failed!") end)
        end
    end)
end

-- Zeit von einem NTP-Server alle Stunde holen
read_ntp()
tmr.alarm(0, 3600000, 1, function() read_ntp() end)

-- UNIX-Zeit jede Sekunde ausgeben
trm.alarm(1, 1000, 1, function() print(rtctime.get()) end)
```

Netzwerkclient

```
-- IP/Port des Servers
svr_ip, svr_port = "10.1.1.82", 8266

-- Verbindung zu Server aufbauen, Daten anfordern und empfangen
function request_svr ()
    local conn = net.createConnection(net.TCP, 0)
    conn:on("receive", function(conn, c)
                           print("\nreceive..."..c.."")
                           do_something(c)
                           conn:close()
                           conn=nil
                       end)
    conn:on("connection",  function()
                               conn:send("get_weather_current\n")
                           end)
    conn:connect(svr_port, svr_ip)
end

-- Daten holen...(alle 60s)
tmr.alarm(1, 60000, 1, function() request_svr() end)
```

MQTT-Client

```
-- Initialisierung (ClientID, Keepalive)
m = mqtt.Client("clientid", 120)
-- ...wenn mit Broker verbunden
m:on("connect", function(client) print ("connected") end)
-- ...wenn Broker offline
m:on("offline", function(client) print ("offline") end)
-- ...wenn eine Nachricht kommt
m:on("message", function(client, topic, data)
                  print(topic .. ":" )
                  if data ~= nil then print(data) end
                end)
-- mit Broker verbinden
m:connect("10.1.1.82", 1883, 0,
                  function()
                      tmr.alarm(0, 1000, 1, function()
                                            m:publish("/topic","Hallo",0,0)
                                            end)
                  end,
                  function() print("connect failed") end)
-- Topic abonnieren
m:subscribe("/topic",0 , function(client) print("subscribe success") end)
```

Accesspoint

```
-- ...keine init.lua vorher...

-- MAC-Adresse des (zukuenftigen) AccessPoints ausgeben
print("MAC:"..wifi.ap.getmac())

-- AccessPoint konfigurieren/starten
wifi.setmode(wifi.SOFTAP)
wifi.ap.config({ssid="esp8266", pwd="espespesp", auth=wifi.OPEN})
wifi.ap.setip({ip="10.1.2.1", netmask="255.255.255.0", gateway="10.1.2.1"})

-- DHCP konfigurieren/starten
wifi.ap.dhcp.config({start="10.1.2.2"})
wifi.ap.dhcp.start()

-- Liste der angemeld. Clients am AP alle 3 Sekunden
tmr.alarm(0, 3000, 1, function()
                          for mac,ip in pairs(wifi.ap.getclient()) do
                              print(mac,ip)
                          end
                      end)
```

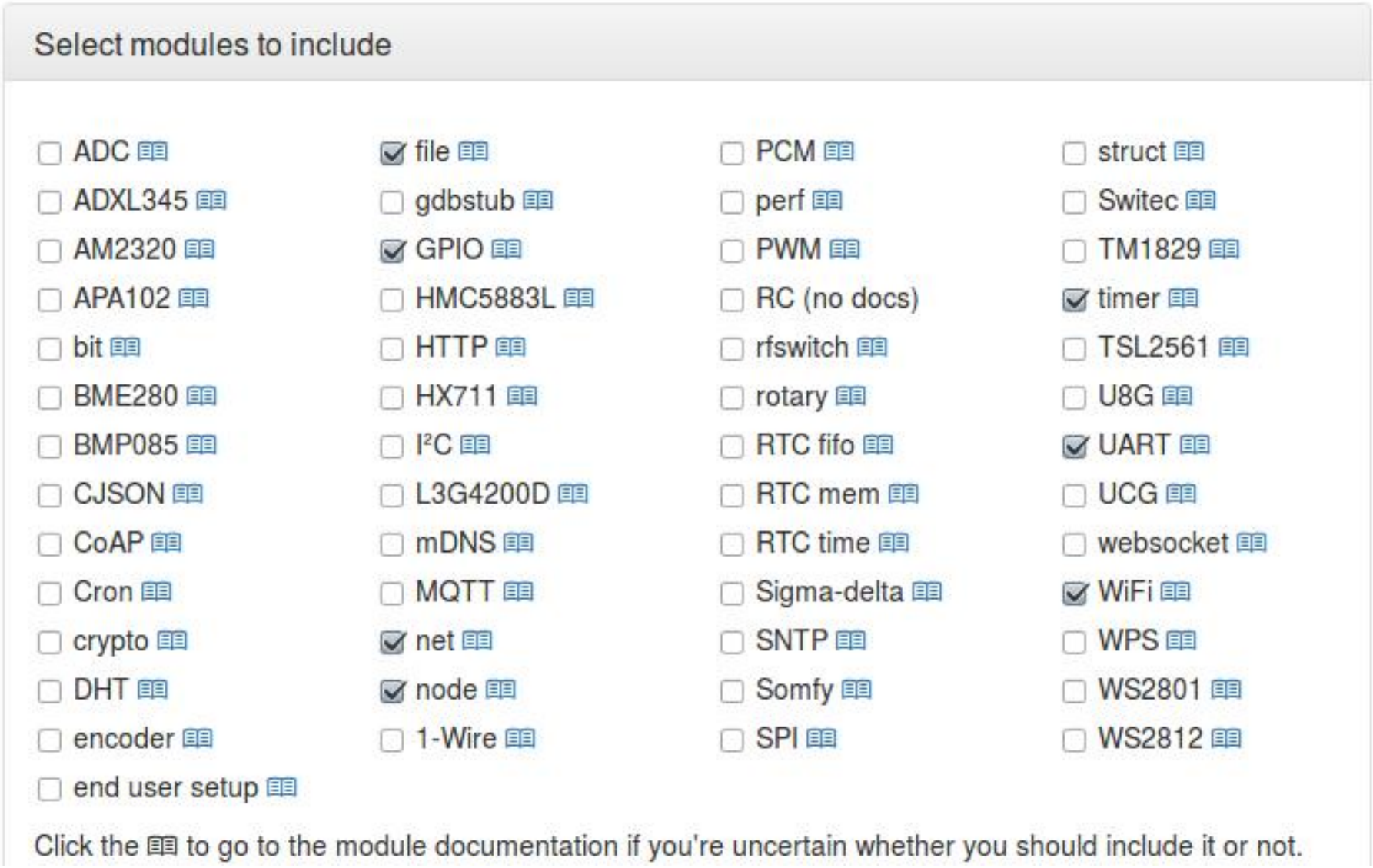
Live-Demo?

Wifi-Scanner

```
-- ...keine init.lua vorher...

--Wifi-Mode STATION, um SSID-Broadcast empfangen zu koennen
wifi.setmode(wifi.STATION)

-- Callback fuer wifi.sta.getap(): Liste der gefundenen APs ausgeben
function listap(t)
    for ssid,v in pairs(t) do
        local auth, rssi, bssid, ch =
              string.match(v, "([^,]+),([^,]+),([^,]+),([^,]+)")
        print(ssid..", "..bssid..", "..rssi.."db, "..auth..", "..ch)
    end
    print("")
End

-- Scan-Filter
scan_cfg = {ssid=nil, bssid=nil, channel=0, show_hidden=1}

-- jede 3 Sekunden Liste der gefundenen WLAN-APs ermitteln/ausgeben
tmr.alarm(0,3000,1,function() wifi.sta.getap(scan_cfg, 1, listap) end)
```

Live-Demo?

NodeMCU-Module

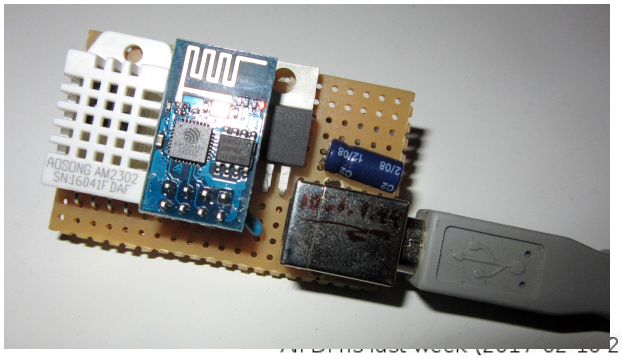


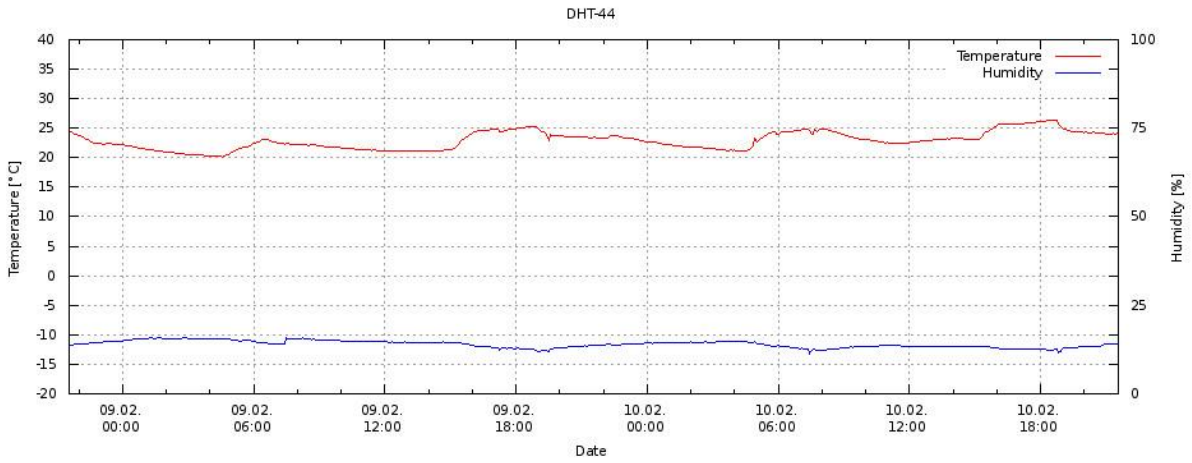
print(node.heap())

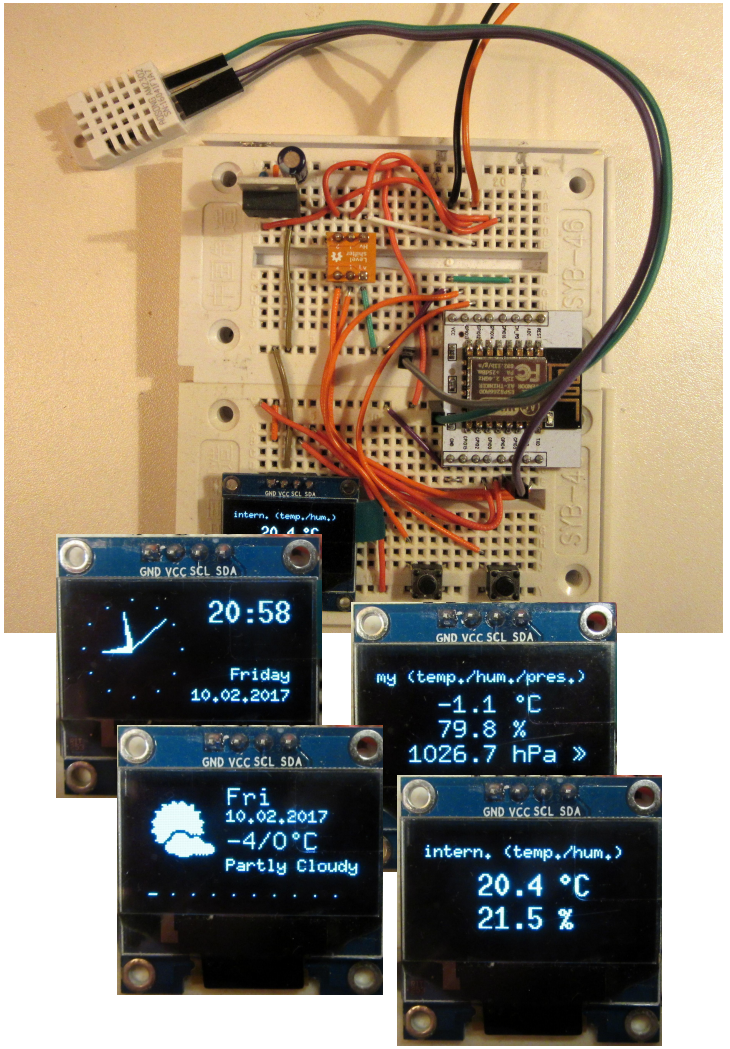
- `node.heap()` gibt den freien Speicher für dynamische Daten aus...
- Um Lua-Funktion `collectgarbage()` eine Chance zu geben:
  - Lokale Variablen und Funktionen, wenn möglich
  - Lua-Module herausschmeißen, wenn sie nicht mehr benötigt werden
  - Lua-Scripte/-Module gleich in Bytecode übersetzen und im Filesystem ablegen
  - ...?

```
--- ...wenn alles nicht hilf, dann harte Methode:
if node.heap < 10000 then
    node.restart()
end
```

Zwei meiner Projekte in Bildern...







...und abseits von NodeMCU?

- Originalfirmware: „lustige“ AT-Befehle
- C mit NON-OS SDK von Espressif
- Arduino core for ESP8266
- Micropython
- smartJS
- ESP8266 Basic
- ESP8266Forth
- ...
- → http://www.mikrocontroller.net/articles/ESP8266#Benutzung_einer_Firmware

Linksammlung

- Toolchains:
  → http://frightanic.com/iot/tools-ides-nodemcu/
- Cloud Build:
  → https://nodemcu-build.com/
- NodeMCU manual:
  → https://nodemcu.readthedocs.io/en/master/
- Lua allgemein:
  → http://lua.org
- Meine NodeMCU-Programme:
  → https://github.com/boerge42/nodemcu_scripts

Ende!

Backup...

JSON-Daten holen/verarbeiten

```
{"query":
    {"count":1,
     "created":"2016-11-25T21:23:23Z",
     "lang":"de",
     "results":
        {"channel":
            {"units":{...},
             "title":"Yahoo! Weather - Brandenburg, BB, DE", ...,
             "location":{"city":"Brandenburg","country":"Germany", ...},
             "wind":{"chill":"34","direction":"90","speed":"6.44"},
             "atmosphere":{"humidity":"93", ...},
             "astronomy":{"sunrise":"7:51 am","sunset":"4:3 pm"},
             "image":{},
             "item":{"title":"","lat":"52.408489","long":"12.56256", ...,
                     "condition":{},
                     "forecast":[{},{},{},{},{},{},{},{},{},{}],
                     "description":"...",
                     "guid":{"isPermaLink":"false"}
                   }
              }
            }
        }
}
```

JSON-Daten holen/verarbeiten

```
local data={con={}, atm={}, wind={}, astro={}}

-- Wettervorhersage holen
function get_weather_from_yahoo ()
    -- Brandenburg an der Havel
    local woeid = 640720
    local url = "http://query.yahooapis.com/v1/public/"..
                "yql?q=select%20*%20from%20%20weather.forecast%20where"..
                "%20woeid%20%3D%20"..woeid.."&format=json"
    local json_data
    -- Wetterdaten holen
    http.get(url, nil, function(code, json_data)
        if (code < 0) then
            print("HTTP request failed")
        else
            -- JSON-Daten dekodieren
            local d = cjson.decode(json_data)
            data.atm = d.query.results.channel.atmosphere
            data.wind = d.query.results.channel.wind
            data.astro = d.query.results.channel.astronomy
        end
    end)
end
```

TCP ↔ UART

```
-- serielle Schnittstelle 0 konfigurieren
uart.setup(0,9600,8,0,1,0)

-- TCP-Server auf Port 9999
global_c = nil
sv=net.createServer(net.TCP)
sv:listen(9999,    function(c)
                       -- eventeull vorherige Verbindung schliessen
                       if global_c~=nil then  global_c:close() end
                       global_c=c
                       -- Daten via TCP empfangen und seriell senden
                       c:on("receive",function(sck, data)
                                      uart.write(0, data)
                                      end)
                   end)

-- wenn Daten ueber seriell kommen, dann diese via TCP senden
uart.on("data", 4, function(data)
                       if global_c~=nil then global_c:send(data) end
                   end,
        0)
```

Tasten abfragen

```
-- wird aufgerufen, wenn Taster geschlossen wird
function switch_down()
    gpio.trig(pin, "none")
    tmr.alarm(0, debounce_delay, tmr.ALARM_SINGLE, function()
                                        gpio.trig(pin, "up", switch_up)
                                        do_something()
                                        end)
end

-- wird aufgerufen wenn Taster geoeffnet wird
function switch_up()
    gpio.trig(pin, "none")
    tmr.alarm(0, debounce_delay, tmr.ALARM_SINGLE, function()
                                        gpio.trig(pin, "down", switch_down)
                                        end)
end

-- Taster konfigurieren
pin, debounce_delay = 6, 30
gpio.mode(pin, gpio.INT, gpio.PULLUP)
gpio.trig(pin, "down", switch_down)
```