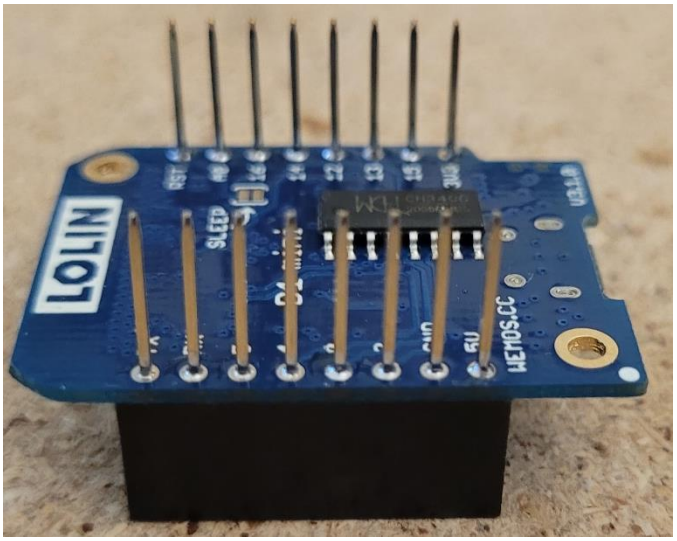
Übung 6

1. Gelöteter Wemos:



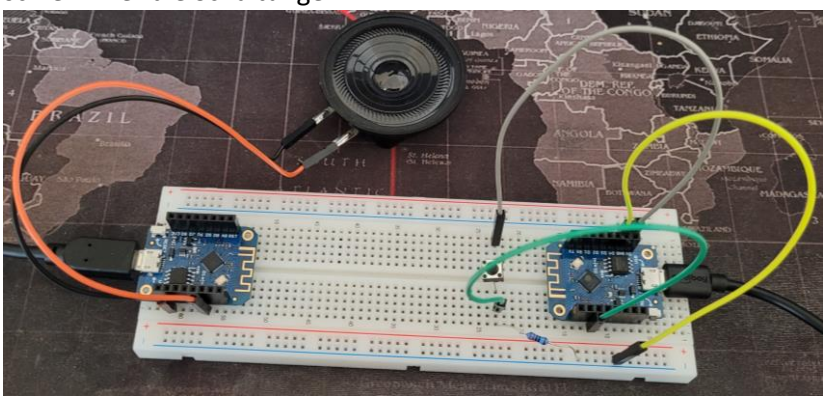
1.1. Anfangs musste erstmal ein neues Projekt mit dem neuen Board angelegt werden bzw. die Platformio.ini entsprechend geändert werden. Es gab nicht direkt das „Wemos D1 Mini“-Board, sondern nur einige weitere Varianten. Bei meiner Version (V3.1.0) handelt es sich um ein ESP-12 Wifi-Board. Laut Dokumentation ist der Chip ausschlaggebend, sodass ich mich für die „Wemos D1 Mini Pro“-Konfiguration entschieden habe, die auch auf Anhieb funktioniert hat. Anschließend habe ich das DHT-Shield entsprechend des Examples draufgesteckt und den Code implementiert, doch hat der Upload nicht funktioniert. Nach ein wenig Herumprobieren ist mir aufgefallen, dass das Gerät im Geräte-Manager als Unbekannt hinterlegt ist. Also habe ich mich auf die Suche nach einem Treiber gemacht, den ich schnell gefunden habe und so das Problem direkt beheben konnte. Hier das gewünschte Ergebnis:

```
--- Miniterm on COM19  9600,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Humidity: 41.00 %       Temperature: 25.60 *C 78.08 *F
Humidity: 41.00 %       Temperature: 25.70 *C 78.26 *F
Humidity: 41.00 %       Temperature: 25.70 *C 78.26 *F
Humidity: 41.00 %       Temperature: 25.70 *C 78.26 *F
Humidity: 40.00 %       Temperature: 25.80 *C 78.44 *F
Humidity: 40.00 %       Temperature: 25.80 *C 78.44 *F
Humidity: 40.00 %       Temperature: 25.90 *C 78.62 *F
```

1.2. Da ich bisher quasi keine Erfahrung mit Python hatte, musste ich zuerst die entsprechenden Softwarepakete herunterladen. Neben Python selber habe ich außerdem die Python-Umgebung für VSCode (da ich hier auch PlatformIO nutze) sowie den passenden Linter installiert. Der Einfachheit halber habe ich VSCode auch direkt als RunTime-Umgebung für Python definiert, sodass alle Programme direkt hier ausgeführt werden.

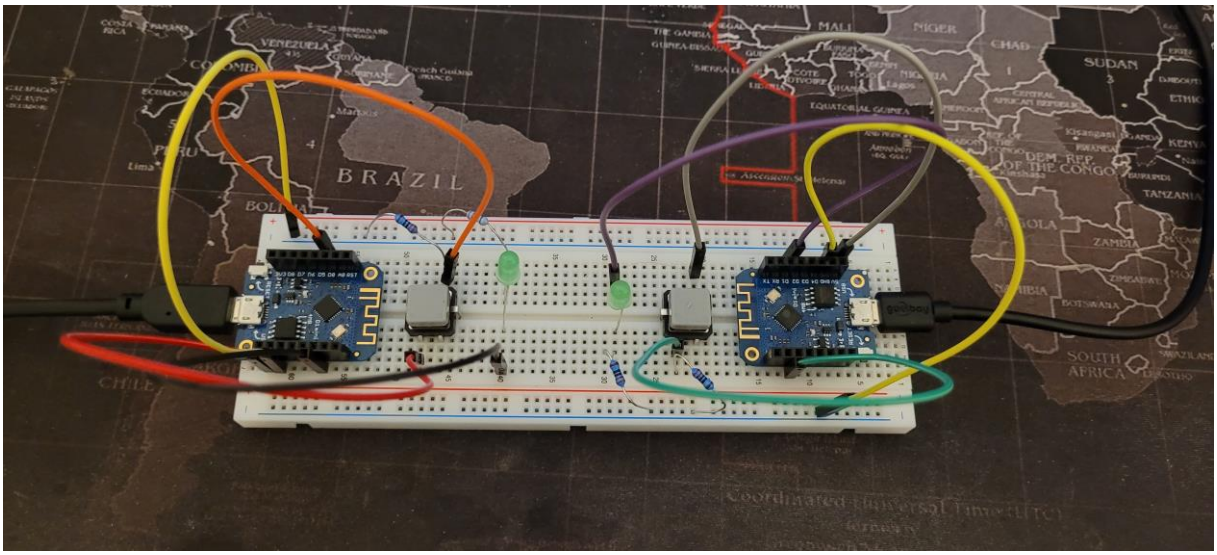
Die Aufgabe selber war eher wie ein „Hello World"-Programm mit Wifi-Kommunikation. Den Code habe ich, mit ein paar Anpassungen, direkt aus den vorgegeben Beispielen genommen und es hat sofort problemlos funktioniert. Bereits oft gehört, durch die fehlende Erfahrung aber nie selber benutzt, habe ich mir „Matplotlib", ***DIE LIBRARY*** zur Datenvisualisierung, ausgesucht. Nach dem ich mich in die Basics eingelesen habe und meine „ersten Visualisierungen" gemacht habe, habe ich mich an der Umsetzung einer Real-Time-Visualisierung versucht. Dazu habe ich die animate-Function genutzt, die genau das macht, was ich mir vorgestellt habe. Allerdings sind so ziemliche alle Beispiele mit lokalen Daten. Meine Herausforderung: Zur Datenabfrage wurde eine while(TRUE)-Endlosschleife benutzt und die FuncAnimation ist eine eigene Schleife. Nachdem ich einige Optionen (Verweise, globale Variablen, Funktionsaufrufe in der while, etc.) versucht habe, habe ich mir die Dokumentation nochmal etwas genauer angeschaut. Die in den Parametern übergebene Funktion func wird zu Beginn jeden Intervalls ausgeführt. Die Lösung: die FuncAnimation ersetzt die while(true)-Schleife. So einfach die Lösung doch ist, solange habe ich gebraucht, um tatsächlich drauf zu kommen. Letztendlich habe ich die beiden Werte in zwei verschiedenen Plots übereinandergelegt, wo man die Entwicklung der Werte sehen kann. Der Code und das passende Video sind im entsprechenden Ordner zu finden.

2. Auch hier habe ich den Code für Server und Client direkt aus den Beispielen übernommen und kurz um die gewünschten Funktionalitäten ergänzt. Da hier lediglich ein paar andere Anweisungen benötigt wurden, war das kein Problem. Für den Klingel-Sound habe ich eine etwas längere Standard-Melodie gewählt und selber nachgebaut. Da sie immer noch sehr kurz ist, habe ich sie direkt implementiert und nicht aus die bekannten Methoden vorheriger Übungen zurückgegriffen, da die dynamischen Methoden mehr Code generieren.
Die sensiblen Daten wie Wifi-SSID und das Passwort befinden sich in der secret.h und werden nicht mit gepusht.
Die Schaltungen zu bauen war ebenfalls schnell gemacht. Einziges Problem: Beim Wemos sind einige Pins bereits vorbelegt und können nicht direkt benutzt werden. Daher musste ich den „Speaker"-Pin und den „Button"-Pin wechseln. Außerdem habe ich mich beim Speaker gegen das Buzz-Shield entschieden, da die Umsetzung mit dem großen Lautsprecher deutlich einfacher schien. Hier die Schaltungen:



Die entsprechenden Videos und der Code sind im entsprechenden Ordner zu finden.

3. Die Aufgabe an sich ist sehr ähnlich zur Aufgabe 2. Daher habe ich den Basis-Code für Server und Client wiederverwendet. Auch hier sind alle sensiblen Daten in der secret.h hinterlegt. Diesmal ist es allerdings egal, welcher Wemos als Server und welcher als Client fungiert. Um sie tatsächlich austauschbar zu machen, habe ich bei beiden dieselben Pins benutzt und einfach gespiegelt aufgebaut:



Ein kleines Problem hat sich allerdings ergeben: Bei meinem Breadboard ist die Reihe 42 auf einer Seite nicht funktionsfähig. Bei den ersten Tests meines Codes hat ein Button immer funktioniert, der andere nicht. Nachdem ich die komplette Schaltung neu gebaut habe und beim Code eine einfache Schaltung ohne Wifi gebaut habe und immer noch keinen Fehler gefunden habe, ist mir die beschädigte Reihe eingefallen, die mir endlich die Lösung gebracht habe. Danach hat alles ziemlich reibungslos funktioniert.
Da sehr wenig Daten gesendet werden, habe ich mich für das dauerhafte Senden der Button-States entschieden. So wird jede Veränderung direkt gesendet und dargestellt. Bei größeren Datenmengen ist dies keine ideale Lösung, für diesen Zweck generiert es aber den wenigsten Code und funktioniert einwandfrei. Einzige Auffälligkeit: Auf Client-Seite gibt es eine Latenz. Die LED benötigt minimal länger um zu leuchten.

4. Problem 1:
Zu Beginn habe ich ein neues Projekt mit dem M5Stack als Board angelegt. Nachdem ich die passende Library installiert hab und den Code kopiert hab, habe ich bei der „#include <M5Stack.h>“-Anweisung eine Fehlermeldung bekommen, dass ich meinen include-Path updaten soll, weil die Quelle nicht gefunden wurde. Irgendwann habe ich in der c_cpp_properties.json die inlcude-Anweisungen gefunden und überprüft. Dabei ist mir aufgefallen, dass ein großer Teil des Packages „framework-arduinoespressif32“, dass mit dem M5-Board installiert wurde, nicht gefunden wurde. Als Lösung wollte ich das Package einfach nochmal neu installieren. Nach einigen vergeblichen Versuchen (neues Projekt, Projekt kopieren, PlatformIO Packages überprüfen lassen) habe ich irgendwann das gesamte Package manuell aus dem PlatformIO-Installationsordner gelöscht. Beim Neustart von VSCode wurde dann das fehlende Package festgestellt und automatisch neu installiert.

Problem 2:
Trotz Installation des benötigten Treibers, konnte ich keinen neuen Code auf den M5 uploaden.

Ich habe folgenden Fehler bekommen:

```
Looking for upload port...
Auto-detected: COM18
Uploading .pio\build\m5stack-fire\firmware.bin
esptool.py v3.0
Serial port COM18
Connecting.....__
Chip is ESP32-D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 84:0d:8e:25:9d:00
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Warning: Could not auto-detect Flash size (FlashID=0xffffff, SizeID=0xff), defaulting to 4MB
Compressed 17104 bytes to 11191...

A fatal error occurred: Timed out waiting for packet content
*** [upload] Error 2
========================================================================================== [FAILED] Took 10.73 seconds ===
The terminal process "C:\Users\Maxi\.platformio\penv\Scripts\platformio.exe 'run', '--target', 'upload'" terminated with exit code: 1.

Terminal will be reused by tasks, press any key to close it.
```

Irgendwann hat es irgendwann dann einigen Versuchen zufällig doch geklappt. Nach ein paar weiteren Versuchen und etwas Recherche hab ich dann diese LÖSUNG gefunden. Anscheinend geht mein ESP32 nicht automatisch in uploading mode, sondern ich muss das manuell machen. Seltsamerweise muss ich das seit der ersten Fehlerbehebung nicht bei jedem Code-Upload machen, sondern der Fehler tritt zufällig auf. Vielleicht gibt es hier einen anderen Grund/ Fehler, aber die Lösung funktioniert dennoch.
Die erfolgreiche Umsetzung des „Hello World“-Programms hat ca. 2,5 Stunden gedauert!

Update zum Upload-Problem: Irgendwann ist der M5 einfach ausgegangen. Auch wenn mit USB verbunden. Nach erneutem einschalten, ist er direkt wieder ausgegangen. Also hatte ich irgendein Problem mit der PSU in Verdacht. Nachdem ich den M5 nicht an den USB-Hub angeschlossen habe, sondern direkt an einen 3.0 USB-Port, waren alle Probleme gelöst. Scheint als möge mein M5 den Hub nicht.

Update 2: Die Fehlermeldungen bekomme ich immer noch, aber deutlich seltener. USB-Kabel rein/raus und VSCode neustarten und es funktioniert alles. Immer noch keine Ahnung, was genau das Problem ist, aber es passiert sehr sehr selten und ich habe eine zuverlässige Lösung parat.

4.1. Nachdem ich obige Probleme endlich behoben hatte, hat alles dann problemlos funktioniert:

4.2. Nachdem die ersten Probleme behoben waren, war der Rest der Aufgabe kein Problem. Bei der Umsetzung habe ich mich an den originalen EXAMPLES des M5 orientiert. Insbesondere an den Display-Beispielen in „Basics“ und „Advanced“. Außerdem habe ich mir die „Games“-Beispiele angeguckt, bei denen Bildschirm-Updates und kontinuierliche Positionierung deutlich besser behandelt werden.
Letztendlich fehlte nur noch die Umrechnung des Millisekunden-Wertes in einfach lesbare Zeit , die ich nach diesem TUTORIAL adaptiert habe. Passender Code und Video sind im Ordner zu finden.

4.3. Auch hier habe ich mich an den Standard-Examples orientiert, insbesondere am ACC-EXAMPLE. Die Werte auszulesen war kein Problem und bei einer „normalen“ Wasserwaage nur eine Achse betrachtet wird, brauchen wir hier nur einen Wert. Durch Anzeigen der Werte auf dem Bildschirm und ausprobieren, erschien mir für eine intuitive Nutzung der X-Wert am besten.
Die Herausforderung bei dieser Aufgabe war dann, die Anzeige des Kreises korrekt hinzubekommen. Da die Werte des Accelerometers zwischen -1 und 1 liegen und als Float mit zwei Nachkommastellen gespeichert werden, gibt es 200 potentielle Rückgabewerte. Diese müssen dann auf die Breite des LCD-Displays gemappt werden. Nach den ersten Versuchen funktionierte nahezu alles, lediglich die ersten und letzten 60 Werte wurden „nicht erreicht“. Zur Berechnung der Koordinaten habe ich das Verhältnis von potentiellen X-Werten zur Bildschirmbreite habe ich breite/200 gerechnet, was einen Integer als Ergebnis hat. Der Multiplikator war dann 1 anstatt 1,6. Die Lösung war relativ einfach: breite/ 200,0. Dadurch wird das Ergebnis ein Float.
Das zweite Problem war das Flickern des Bildschirms. Anfangs wurde bei jeder Aktualisierung der Bildschirm gecleared und dann alles neu gezeichnet, was zu starkem Flickern geführt hat. Um ein neues Zeichnen aller Elemente zu vermeiden und das Flickern zu reduzieren, sollte nur der Kreis neu gezeichnet werden und der alte Kreis „übermalt“. Zusätzlich habe ich ein kurzes delay eingefügt. Ausgehend davon, dass eine Wasserwaage zumeist still ist, reicht eine leicht verlängerte Aktualisierungszeit. Das angenehmste Flickern habe ich bei 50ms delay empfunden. Die Ideen habe ich von HIER adaptiert.
Passender Code und Video sind hier im Ordner zu finden.