EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“	
1.0a Aufgabe	Arduino-IDE 2.3.6 installieren. Die aktuelle Entwicklungsumgebung zum Schreiben von Sketches auf Windows 11 (10) installieren.
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin
Download	Voraussetzung ist die **ZIP-Installation**. https://downloads.arduino.cc/arduino-ide/arduino-ide_2.3.6_Windows_64bit.zip (verwende den USB-Stick)
Installation	1. Kopiere die ZIP-Datei für den Workshop nach: **„> Dieser PC > Boot (C:) > Benutzer > Öffentlich >“** kopieren. (Windows-Pfad lautet: „**C:\Users\Public**“) 2. ZIP-Ordner entpacken. 3. Link zu „**Arduino IDE.exe**“ auf dem Desktop erstellen. **Nicht starten!** 4. Im Installationsordner „**Arduino-IDE2“** den Ordner „**Sketchbook**“ anlegen.
Hinweise	Der Ordner „libraries“ wird im Pfad des jeweiligen Sketchbooks angelegt. Das ist standardmäßig: „C:\Users\<Nutzer>\Arduino\libraries“. Dies gilt nur solange, bis wir in den IDE-Einstellungen den Sketchbook-Pfad ändern.
Empfehlung	Wer nicht bei jedem Projekt die Libraries neu installieren möchte, sollte diesen Pfad nicht wieder ändern.
Erster Start	5. Hier eine Entscheidung treffen: Wer keine portable Version braucht, macht weiter mit Schritt 6, ansonsten Folgeabschnitt „*Portable Installation*“. 6. IDE starten. 7. Im Menü „**Datei**“ die „**Einstellungen**“ anwählen, und den „Pfad für Sketchbook“ verschieben nach „**C:\Users\Public\Arduino-IDE2\Sketchbook**“. 8. Kopiere die Dateien aus dem Stick-Ordner „…\ESP32“ auf den Desktop. Hier finden sich die Workshop-Sketche. 9. Weiter mit der nächsten Aufgabe.
Abschnitt: Portable Installation (Eine manuelle portable Installation wird in Aufgabe „1.0b“ gezeigt)	
Eine portable IDE 2 mit der Batch-Datei „make_arduino-cli.yaml.cmd“ erstellen.	
	10. Kopiere die **Dateien** aus dem Stick-Ordner „…\Copy Dateien nach Arduino-IDE2“ in den Installationsordner der IDE 2.
Mit der Batch-Datei „make_arduino-cli.yaml.cmd“ lokalisieren wir die IDE 2. Das heißt: Der Ordner „…\local\Arduino15“ wird nunmehr im Installationsordner angelegt!	
	11. Klicke auf “make_arduino-cli.yaml.cmd”.
Kontrolle	12. Gehe mit dem Explorer nach: „C:\users\<Nutzer>\.arduinoIDE“ 13. Öffne die Datei „arduino-cli.yaml“ im Editor. 14. Kontrolliere die Pfade
IDE-Start	15. Start die IDE 2 vom Desktop.

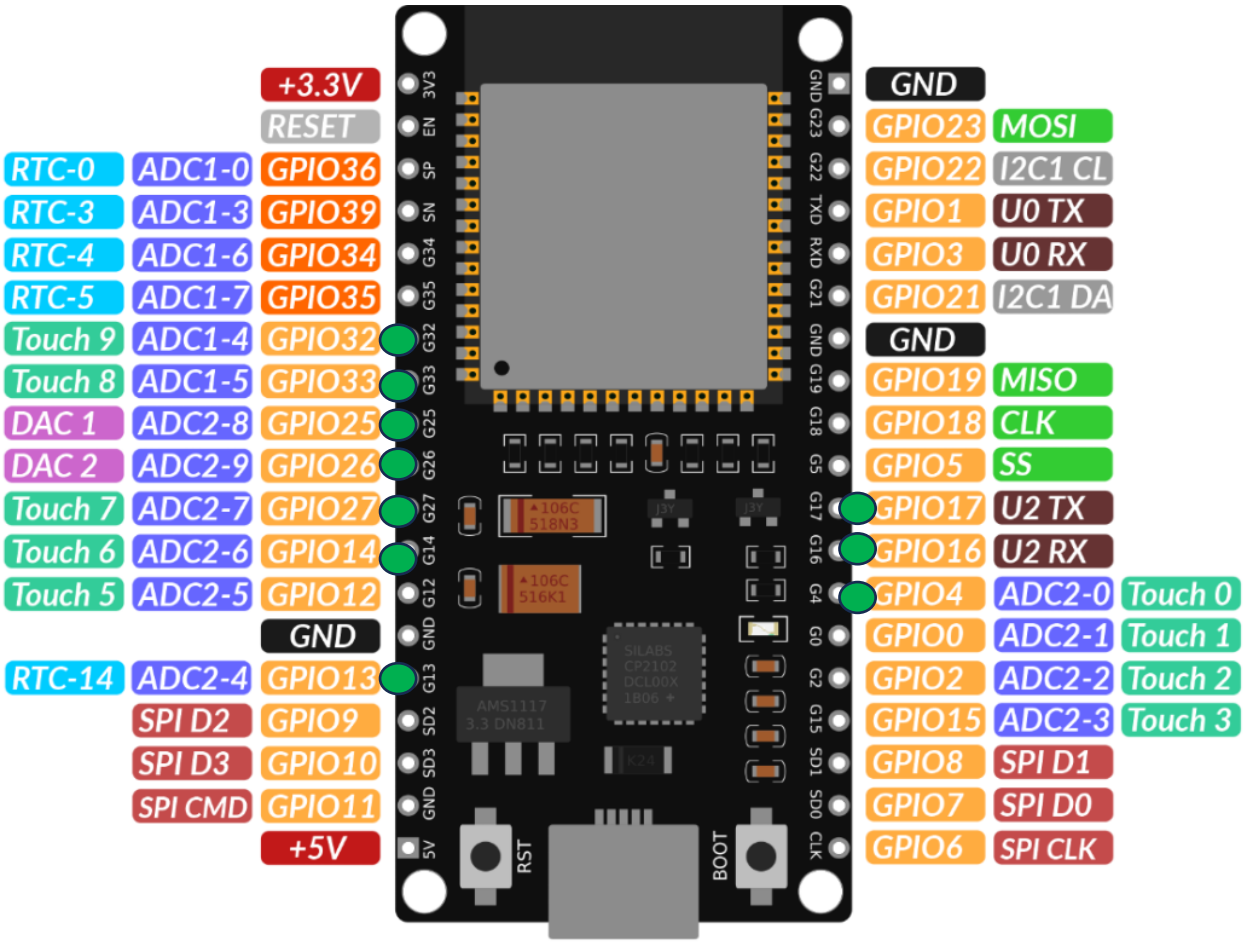
EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“	
1.0b Aufgabe	Zugabe für Experten: Arduino-IDE 2.3.6 manuell portabel installieren. (Ordnerstruktur der IDE 2.3.6 im Detail).
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin
Download	Voraussetzung ist die **ZIP-Installation**. https://downloads.arduino.cc/arduino-ide/arduino-ide_2.3.6_Windows_64bit.zip
Dieser Abschnitt ist ein Bonbon für Fortgeschrittene, die eine portable Version erstellen möchten.	
Installation	IDE wie oben geschildert installieren und starten. Sketchbook-Pfad im Menü „**Datei**“ die „**Einstellungen**“ neu zuordnen.
Portabel?	• Die IDE „2.3.6“ ist als nicht portabel definiert, da während des ersten Starts die heruntergeladenen Dateien unter dem Ordner „**C:\Users\<Benutzer>\Appdata\Local\Arduino15**“ abgelegt werden. • Weiter wird eine Konfigurationsdatei in „**C:\Users\<Benutzer>\.arduinoIDE**“ angelegt. • Auch in „C:\Users**\<Benutzer>\.**AppData\Roaming\arduino-IDE“ werden Einstellungen abgelegt.
Trick	Eine portable Installation kann mit folgenden Schritten erzeugt werden. Alle von der IDE installierten Dateien liegen dann im Installationsordner, hier **„Arduino-IDE2“.**
Appdata verschieben	Den Ordner „…**Appdata\Local\Arduino15**“ nach „**C:\Users\Public\Arduino-IDE2\Arduino15**“ verschieben.
arduino-cli.yaml editieren	Die Konfigurationsdatei liegt in „**C:\Users\<Benutzer>\.arduinoIDE**“. In einem Editor „**arduino-cli.yaml**“ öffnen und editieren.
Pfade ändern	“libraries: c:\Users**Public\Arduino-IDE2**\Arduino15\libraries” “data: c:\Users**Public\Arduino-IDE2**\Arduino15” “user: c:\Users**Public\Arduino-IDE2**\Sketchbook”
arduino-cli.yaml speichern	Konfigurationsdatei “**arduino-cli.yaml**“ speichern. Beim nächsten IDE-Start werden Dateien im Installationsverzeichnis „**C:\Users\Public\Arduino-IDE2\Arduino15**“ abgelegt!
Umzug auf neuen Rechner.	
.arduinoIDE kopieren	Den Ordner „**C:\Users\<Benutzer>\.arduinoIDE**“ nach „**C:\Users\Public\Arduino-IDE2**“ sichern (kopieren).
Installation kopieren	Den Installationsordner „Arduino-IDE2“ auf den neuen Rechner kopieren. (Wer den Installationsort ändert, muss “**arduino-cli.yaml**“ anpassen.
.arduinoIDE verschieben	Den Ordner „**C:\Users\Public\Arduino-IDE2\.arduinoIDE**“ nach „**C:\Users\<Benutzer>**“ verschieben.
Erster Start	Link zu „Arduino IDE.exe“ auf dem Desktop erstellen. IDE erneut starten.

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
2.0 Aufgabe	Sketchbook „ESP32“ Sketchbook einrichten. Das Sketchbook „ESP32“ zum Schreiben der Sketches installieren. Symlink einrichten.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin	
IDE 2 Verhalten	Die IDE 2 legt zunächst in jedem projektbezogenen Sketchbook einen weiteren "libraries-Ordner" an. Damit werden bisher installierte Libraries nicht mehr gefunden. Daher müsste man die bisher installierten Libraries in den neuen Sketchbook Ordner kopieren.	
Abhilfe	Eine Batch-Datei die einen Systemlink auf einen „zentralen Libraries-Ordner erzeugt“.	
MSDOS	Nur zur Information: Syntax: mklink /J "link" "target" "link" enthält den Pfad zum neunen Sketchbook-Ordner. Hier muss der Symlink liegen. "target" ist der Pfad zum zentralen Libraries-Ordner.	
Batch-Datei	Die Batch-Datei muss sich im Ordner des neuen Sketchbooks befinden. Hier: „C:\users\<Nutzer>\Desktop\ESP32“.	
Hinweis	In der Batch-Datei „**MakeSketchbookLink_In_Sketchbook.cmd**“ ist der Pfad auf den zentralen Libraries-Ordner eingestellt auf: “**C:\Users\Public\arduino-IDE2\Sketchbook\libraries”.** Falls Du einen abweichenden Installationsordner hast, musst Du die Datei editieren und hinter „target=“ deinen Installationspfad einstellen.	
Symlink erstellen	16. Führe “MakeSketchbookLink_In_Sketchbook.cmd” aus.	
IDE starten	17. Starte erneut die IDE. 18. Stelle unter „> Datei > Einstellungen“ den „Pfad für Sketchbook“ auf hier „C:\users\<Nutzer>\Desktop\ESP32“ um.	
Geheimer Workaround für Experten!		

	In der Konfigurationsdatei „**arduino-cli.yaml**“ den Eintrag „libraries“ mit einem Pfad zu einem zentralen Ordner ergänzen:
	`board_manager:`
	`  additional_urls: []`
	`directories:`
	`  builtin:`
	`  libraries: C:\Users\Public\arduino-IDE2\libraries`
	`  data: c:\Users\Public\arduino-IDE2\Arduino15`
	`  user: c:\Users\enno_\Desktop\ESP32`
	`locale: de`	

	Dieser Ordner wird von der IDE beim Kompilieren gesehen.	
Nachteil	Libraries werden nicht im Library Manager angezeigt.	
Manuell	Will man Libraries zentral machen, muss man sie vom Sketchbook-Ordner in den zentralen Ordner kopieren bzw. verschieben.	
Achtung	Der Compiler sucht zuletzt im zentralen Ordner nach Libraries.	

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
3.0	Aufgabe	Pinbelegung verstehen.
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin	



Verfügbare Pins:	
Sicher:	**GPIO13, GPIO14, GPIO27, GPIO26, GPIO25, IGPIO33, GPIO32, GPIO4, GPIO16, GPIO17**
I2C:	GPIO21 (SDA) und GPIO22 (SCL)
SPI:	GPIO5 (CS), GPIO18 (CLK), GPIO19 (MISO), GPIO23 (MOSI)
Nur Input:	GPIO35, GPIO34, GPIO39, GPIO36 (ohne Pull-ups oder Pull-downs)
Touch-Sensoren:	GPIO15, GPIO2, GPIO0, GPIO4, usw.
USB seriell:	GPIO3 (RX), GPIO1 (TX)
SPI nicht nutzen:	GPIO6, GPIO7, GPIO8, GPIO11, GPIO10, GPIO9
Strapping:	GPIO15, GPIO2 (LOW), GPIO0, GPIO4, GPIO5 (sonst HIGH)

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
4.0 Aufgabe	Daten ESP-32 Dev Kit C V2. Grenzwerte für Stromstärke und Spannung.	
Bauteile	ESP32 Dev Kit C V2 (Espressif) ESP32 Terminal Adapter 38 Pin	
Gerätename	Arduino UNO	ESP-32 Dev Kit C V2
Prozessor	ATmega328P **8-bit**	ESP32-WROOM-32 **32-bit**
Prozessortakt	**16 MHz**	**240 MHz**
Speicher Programm Arbeitsspeicher	 32 KB Flash 2 KB SRAM 1 KB EPROM	 4 MB (externer SPI-Flash) 520 KB SRAM 448 KB ROM
Betriebsspannung **I/O**	5 V	**3,3 V**
Spannungsversorgung USB extern	 USB 5 V VIN-PIN 7-12 V	 Micro-USB 5 V **5V Pin maximal 5,5 V**
Max. **Stromstärke**	40 mA	**Je GPIO 10 mA** Max. 500 mA
Frequenzbereich	nicht vorhanden	ISM-Band bei 2,4Ghz: 2,412–2,472 GHz
Bluetooth	nicht vorhanden	Classic & LE
Schnittstellen		UART / GPIO / ADC / PWM / SPI / I2C
GPIOs	14 digital 6 analog	34 programmierbare GPIOs Besondere GPIOs: • 5 strapping GPIOs (Bootprozess) • 6 GPIOs nur für INPUT • 6 GPIOs reserviert für externen Flash • …
Nicht vorhanden	nicht vorhanden	GPIO20, GPIO24, ?
PWM	6 PWM-Kanäle Pin 3, 5, 6, 9, 10, 11	16 PWM-Kanäle: Ausnahmen: GPIO 6–11 → reserviert für Flash GPIO 34–39 → nur **Input**, kein PWM
Touch-Sensoren	nicht vorhanden	10 GPIOs
SPI	1 SPI-Controller	4 SPI-Controller, z.B.: MISO=19, MOSI=23, CLK=18, CS=5
I2C	1 I2C-Controller	2 I2C-Controller, z.B.: SDA = GPIO21; SCL = GPIO22
ADC	6 Kanäle	ADC1: 6 Kanäle ADC2: 10 Kanäle (nicht bei WiFi)
DAC	nicht vorhanden	DAC 1 & DAC 2
Quelle: „ESP-32 Dev Kit C V2 Betriebsanleitung.pdf“		
Quelle: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/index.html		

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

5.0 Aufgabe Variablen verstehen.

Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin	

Variablen sind Container zum Speichern von Datenwerten.

In C++ gibt es verschiedene Variablentypen definiert mit Schlüsselwörtern, zum Beispiel:

int:	speichert ganze Zahlen (Integer) ohne Dezimalstellen, z. B. 123 oder -123
double:	speichert Gleitkommazahlen mit Dezimalstellen, z. B. 19,99 oder -19,99
char:	speichert einzelne Zeichen, z. B. 'a' oder 'b'. Char-Werte werden in einfache Anführungszeichen gesetzt.
String:	speichert Text, z. B. "Hello World". String-Werte werden in doppelte Anführungszeichen gesetzt.
bool:	speichert Werte mit zwei Zuständen: true oder false, „1“ oder „0“

Variable definieren:

Definition	int varX;	// kein Startwert
Initialisierung	int varX = 5;	// mit Startwert

Tabelle $\mathbf{2^n}$

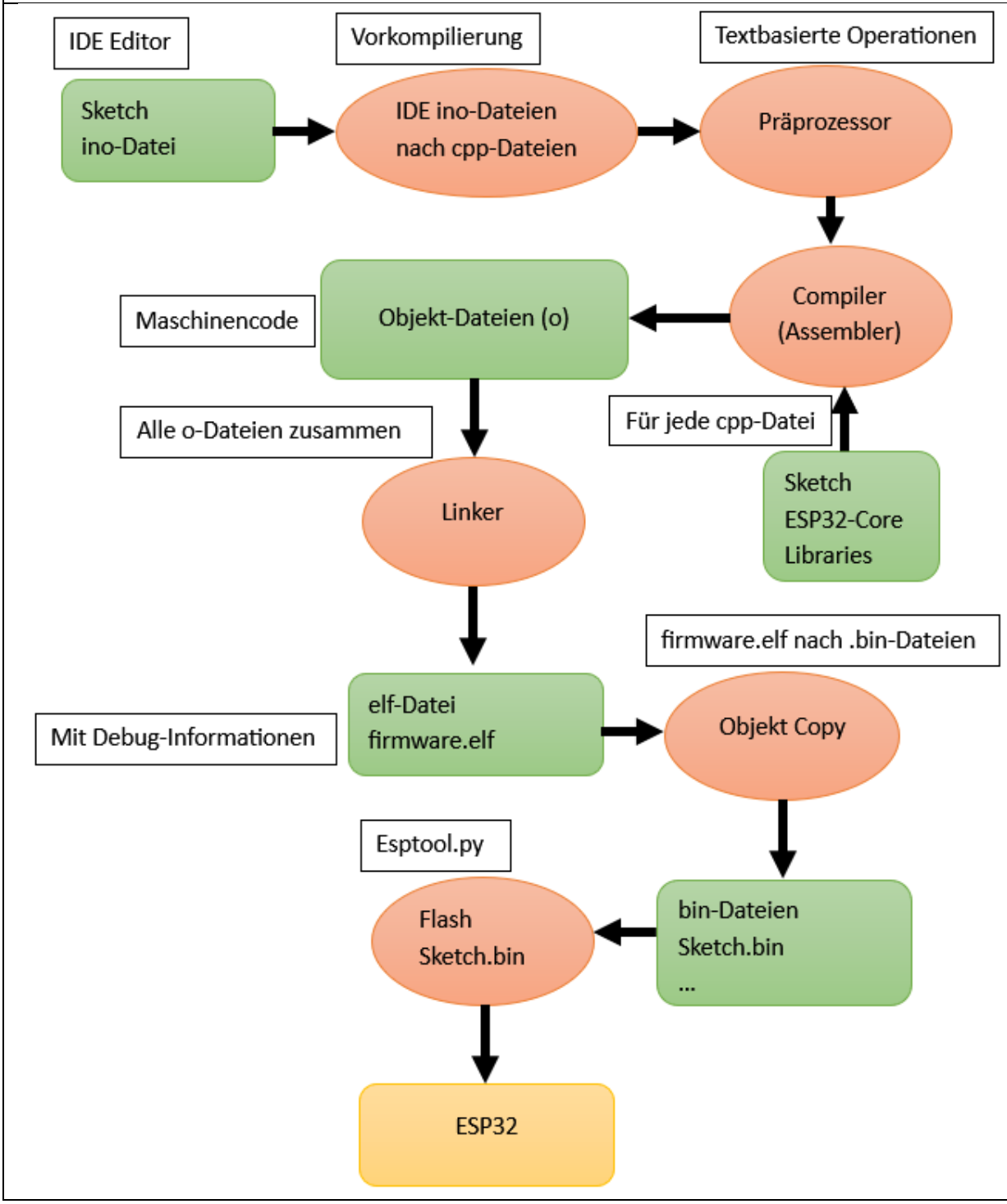
unsigned char	2^8	=	256	(256 - 1 = 255)
unsigned short	2^{16}	=	65.536	(65.536 - 1 = 65.535)
unsigned long	2^{32}	=	4.294.967.296	(4.294.967.296 – 1 = 4.294.967.295)

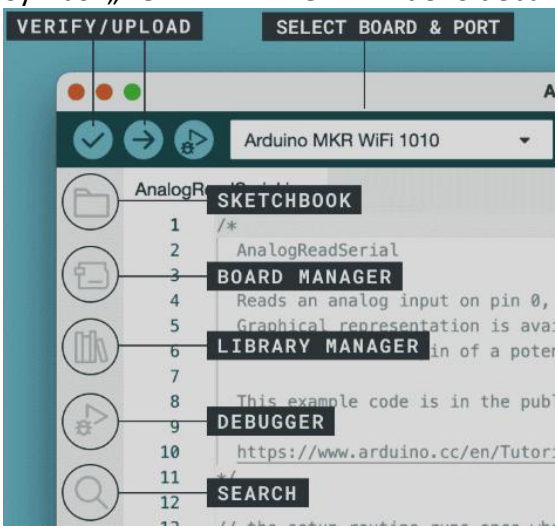
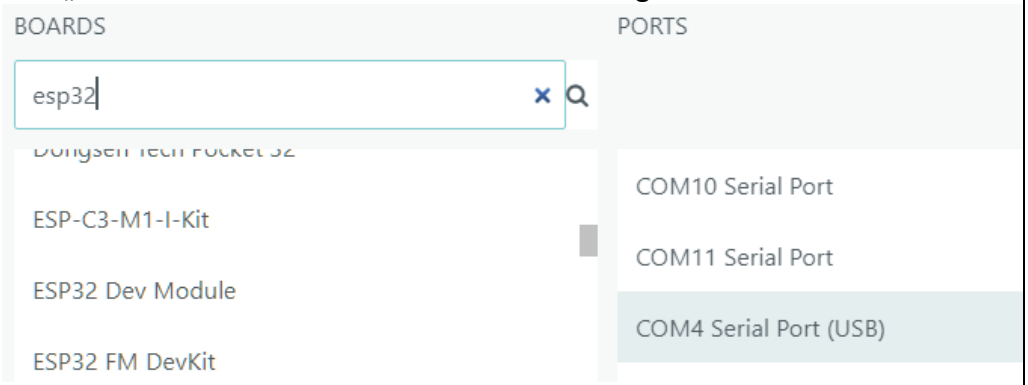
Datentypen	**Wertebereiche**	**ISO/IEC-Konvention**
bool	true oder false, „1“ oder „0“	
char	-128 bis +127	int8_t (1 Byte)
unsigned char	0 bis 255	uint8_t (1 Byte)
short	−32.768 bis +32.767 (beim Arduino int)	int16_t (2 Byte)
unsigned short	0 bis 65.535 (beim Arduino unsigned int)	uint16_t (2 Byte)
int	Beim ESP32 wie „long“	int32_t (4 Byte)
unsigned int	Beim ESP32 wie „unsigned long“	uint32_t (4 Byte)
long	−2,147,483,648 bis +2,147,483,647	int32_t (4 Byte)
unsigned long	0 bis 4,294,967,295	int32_t (8 Byte)
float	$\pm 3.4 \times 10^{38}$	(4 Byte)
double	$\pm 1.7 \times 10^{308}$	(8 Byte)

Zeichenketten	
String-**Klasse**	(kein Datentyp)
String	**S**tring str4 = "Arduino";
String	**S**tring str5 = "Sensor-Wert: "+ analogRead(A0);
Datentyp	
char	char myChar = ‘A’;
char Bezeichner[]	char myChar[] = { ‘E’, ‘S’, ‘P. ‘3’, ‘2’, ‘\0’};
	char myChar[6] = { ‘E’, ‘S’, ‘P. ‘3’, ‘2’, ‘\0’};
Arrays	
Datentyp Bezeichner[]	int myPins[4];
	int myPins[] = { 13, 14, 27, 26 };
	int myPins[4] = { 13, 14, 27, 26 };
Achtung	Das erste Element hat den Index 0!

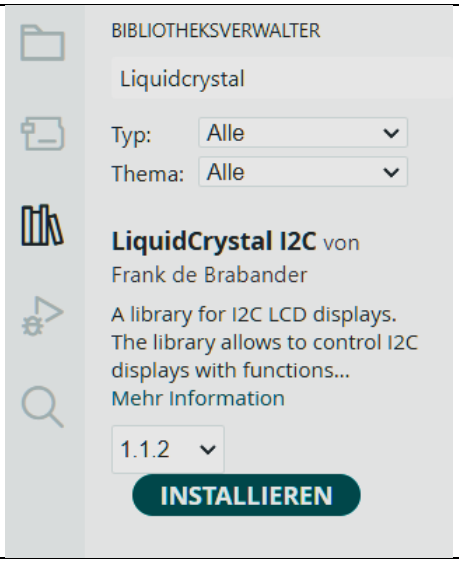
EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
6.0 Aufgabe	ESP32-Anweisungen (statements).	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin	
ino-Struktur	setup() loop()	Wird einmal beim Start ausgeführt. Wird endlos wiederholt.
I/O	pinMode(pin, mode) digitalWrite(GPIO, HIGH/LOW) digitalRead(GPIO) analogRead(GPIO) analogWrite(GPIO, value)	INPUT, OUTPUT, INPUT_PULLUP, INPUT_PULLDOWN GPIO HIGH oder LOW setzen Digitalen Eingang lesen Analogen Eingang (ADC) lesen PWM-Ausgabe
Zeit	delay(ms) delayMicroseconds(µs) millis() micros()	Warten (blockierend) in ms Warten (blockierend) in µs Zeit seit Start in ms Zeit seit Start in µs
Serial	Serial.begin(115200) Serial.print(…) Serial.println(…) Serial.available() Serial.read() Serial.readStringUntil(…) map(x, in_min, in_max, out_min, out_max) constrain(x, a, b)	Serielle Kommunikation (Serieller Monitor) Serielle Ausgabe ohne LF Serielle Ausgabe mit LF Sind Bytes im Eingangspuffer? Byte aus Eingangspuffer lesen String aus Eingangspuffer lesen Wertebereich umrechnen Bereich begrenzen
Bluetooth	#include <BluetoothSerial.h> BluetoothSerial SerialBT SerialBT.begin("EBW-Workshop") SerialBT.available() SerialBT.write() SerialBT.read() SerialBT.readStringUntil(…)	Library einbinden Objekt SerialBT erzeugen SerialBT initialisieren Sind Bytes im Eingangspuffer? Bluetooth Ausgabe Byte aus Eingangspuffer lesen String aus Eingangspuffer lesen
Touch Sensor	touchRead(Touch-GPIO)	Lesen eines Touch Sensors
WLAN		
und mehr…		

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
7.0 Aufgabe	Vom Sketch zum Flashen des ESP32.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin	



EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“	
8.0 Aufgabe	ESP32 Board installieren. Mit Hilfe des „BOARD MANAGERS“ Board hinzufügen.
Bauteile	ESP32 Dev Kit C V2 (Espressif) ESP32 Terminal Adapter 38 Pin
Arduino IDE 2	Symbol „BOARD NANAGER“ in der Sidebar. 
Espressif Board	Auf das Symbol „BOARD MANAGER“ klicken. Suchen nach „ESP32“. Auf „**INSTALLIEREN**“ klicken. 
ESP32 USB	Das ESP32-Board über USB mit dem PC verbinden.
Reiter **SELECT BOARD**	
Board & Port auswählen	Zuerst den Port wählen. Dann im Suchfeld „ESP32“ eintragen. Hier „ESP32 Dev Module“ auswählen und bestätigen. 
Quelle	Getting Started with Arduino IDE 2 | Arduino Documentation

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“	
9.0 Aufgabe	Arduino IDE 2 kennen lernen.
Bauteile	ESP32 Dev Kit C V2 (Espressif) ESP32 Terminal Adapter 38 Pin
Neue Funktionen	Autovervollständigung, ein integrierter Debugger und die Synchronisierung von Sketches mit der Arduino Cloud.
SideBar	Die Arduino IDE 2 verfügt über eine neue Seitenleiste, die den Zugriff auf die am häufigsten verwendeten Werkzeuge erleichtert.
Arduino IDE 2	
Verify / Upload	Überprüfen/Hochladen – Kompilieren und laden Sie Ihren Code auf Ihr Arduino-Board hoch.
Select Board & Port	Board & Port auswählen – erkannte Arduino-Boards werden hier automatisch zusammen mit der Portnummer angezeigt.
Sketchbook	Hier findest Du die aktuellen Sketches.
Boards Manager	Jedes Board muss der IDE 2 bekannt machen. Hier werden die passenden Pakete heruntergeladen.
Library Manager	Unzählige Bibliotheken (Libraries), die von Arduino und seiner Community entwickelt wurden können hier installiert werden.
Debugger	Testen und debuggen der Sketches in Echtzeit.
Search	Suchen nach Schlüsselwörtern im Code.
Open Serial Monitor	Öffnet den Seriellen Monitor als neuen Tab in der Konsole.
Open Serial Plotter	Öffnet den Seriellen Plotter in einem neuen Fenster.
Tricks	Palette der IDE-Konfiguration: Tastenkürzel: „Strg + ⇧ + P“ Es wird eine Liste der Tastenkürzel angezeigt.
„settings UI“	Gebe im Suchfeld „settings UI“ ein.
Echtzeitdiagnose	Gebe in „Einstellungen suchen“ „realtime“ ein. Setze ein Häkchen in „… diese Option aktiviert ist …“
Minimap	Gebe in „Einstellungen suchen“ „minimap“ ein. Setze ein Häckchen in „Editor › Minimap: Enabled“. Editiere die Breite der Minimap …
Quelle	Getting Started with Arduino IDE 2 | Arduino Documentation

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
10.0 Aufgabe	LCD Library installieren. Mit Hilfe des „LIBRARY MANAGERS“ LCD Bibliothek hinzufügen.	
Bauteile	ESP32 Dev Kit C V2 (Espressif) ESP32 Terminal Adapter 38 Pin Steckbrett	
Arduino IDE 2	Symbol „LIBRARY NANAGER“ in der Sidebar. 	
LCD Library	Auf das Symbol „LIBRARY MANAGER“ klicken.	
		Im Suchfeld „**Liquidcrystal**“ eingeben. Aus der Liste der Bibliotheken die Bibliothek von „**Frank de Brabander**“ auswählen. Auf **INSTALLIEREN** klicken.
Quelle	Getting Started with Arduino IDE 2 | Arduino Documentation	

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

11.0 Aufgabe

Serielle Ausgaben.
Über den „Seriellen Monitor“ sollen Informationen ausgegeben werden.
„Hallo Welt“ soll periodisch ausgegeben werden.

Bauteile

ESP32 Dev Kit C V2
ESP32 Terminal Adapter 38 Pin
Steckbrett

Hinweise

- Nach jedem RESET erscheinen auf dem geöffneten „Seriellen Monitor“ Bootloader-Meldungen, dabei muss die Baudrate des „Seriellen-Monitors“ auf 115200 Baud stehen.
- Diese Bootlader-Meldungen lassen sich nicht einfach Unterdrücken.
- Weiterhin muss im Sketch die Baudrate mit 115200 definiert werden. **Serial.begin(115200).**
- Um Ausgaben bereits in der Setup-Methode fehlerfrei zu ermöglichen, sollte die Bereitschaft der seriellen Schnittstelle abgewartet werden. **while(!Serial) {}.**

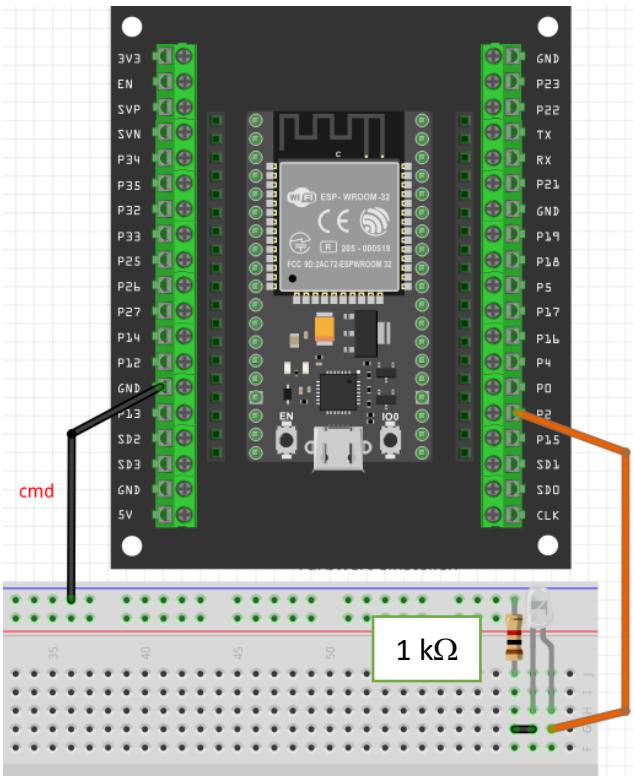
Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, Serial_Print.ino

```
#define SPEED_SERIAL 115200

void setup() {
  // put your setup code here, to run once:
  Serial.begin(SPEED_SERIAL);
  while(!Serial) {}          // wait for established serial connection
  Serial.println("EBW-Workshop");
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.println("Hallo Welt");
  delay(500);
}
```

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
12.0 Aufgabe	Kapazitiver Touch Sensor und serielle Ausgabe.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin Steckbrett 1 x Jumper-Kabel	
Hinweise	Der ESP32 hat 10 interne kapazitive Touch-Sensoren. Die kapazitiven Touch Pins können auch verwendet werden, um den ESP32 aus dem Tiefschlaf aufzuwecken. Die internen Touch-Sensoren sind mit diesen GPIOs verbunden:	
GPIOs	Touch 0 (GPIO 4), Touch 1 (GPIO 0), Touch 2 (GPIO 2), Touch 3 (GPIO 15), Touch 4 (GPIO 13), Touch 5(GPIO 12), Touch 6 (GPIO 14), Touch 7 (GPIO 27), Touch 8 (GPIO 33), Touch 9 (GPIO 32).	
Eigenschaften	Beim ESP32 Touch-Sensor ist der Wertebereich nicht absolut festgelegt. Typischer Wertebereich: Rohwerte ca. 0 bis etwa 1000. Im **unberührten Zustand** liegen die Werte meist zwischen **300 und 800.** Bei **Berührung** fällt der Wert typischerweise auf **50–300.**	
Code	`touchRead(touchPin)`	
	Die ESP32-Funktion `touchRead()` hat keinen definierten Wertebereich, daher Datentyp Integer.	
Schaltung	Verbinde mit GPIO15 (Touch 3) ein Jumper-Kabel (männlich-männlich).	
Versuch	Nehme das Jumper-Kabel zwischen die Finger und drücke mit unterschiedlicher Intensität.	

Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, Serial_Touch.ino

```
const int touchPin = 15;     // GPIO15 defined as Touch 3

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  while(!Serial) {}          // wait for established serial connection
  Serial.println("EBW-Workshop");
}

void loop() {
  // put your main code here, to run repeatedly:

  Serial.print("Touch-Sensor: ");
  Serial.println(touchRead(touchPin));   // predifined function touchRead()
  delay (500);
}
```

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
13.0 Aufgabe	Eine Low Current-LED soll Blinken.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin Steckbrett	LED Low Current (V_F = **1,8 V** I_F = **2 mA**) 1 kΩ Widerstand Jumper-Kabel
Hinweise **Empfehlung**	Als Transformator kommt ein Schaltnetzteil mit max. 9 V infrage. Der Spannungsregler sorgt für 5 V Versorgungsspannung. Der ESP32 dient nicht zur Spannungsversorgung. Laut ChatGPT werden 10 mA für eine längere Lebensdauer empfohlen. LEDs nicht direkt am GPIO betreiben (Ausnahme Low Current LED).	
Achtung	Die GPIOs sollten prinzipiell nicht mit Strömen größer als 10 mA belastet werden.	
Schaltung		

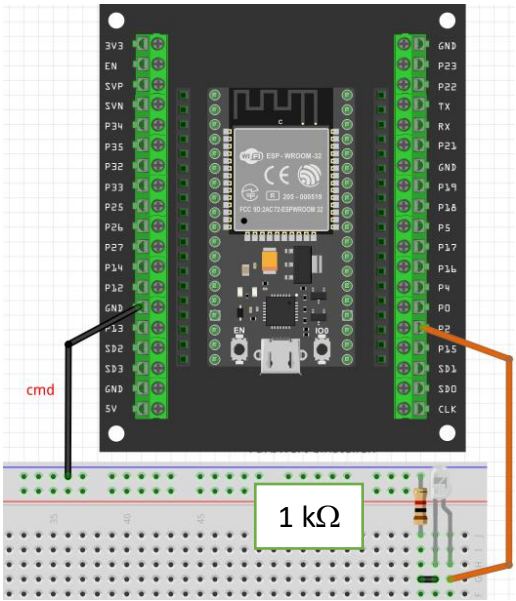
Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, Blink.ino

```
#define LED_BUILTIN 2 // Must be new defined, ESP32 comes without definition

void setup() {
  Serial.begin(115200);
  while(!Serial) {}
  Serial.println("EBW-Workshop");
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the 3.3 V
voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage
low
  delay(1000);                      // wait for a second
}
```

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

14.0 Aufgabe	Eine Low Current-LED soll blinken. Code Variation: **Keine blockierende delay()-Funktion**.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin Steckbrett	LED Low Current (V_F = **1,8 V** I_F = **2 mA**) 1 kΩ Widerstand Jumper-Kabel
Problem	In der Regel wird die delay()-Funktion verwendet, um z.B. ein Blink-Interval zu erzeugen.	
Aufgabe	Den Code so abwandeln, dass eine **nicht-blockierende Zeitsteuerung** erreicht wird. Dies wird erreicht, in dem die millis()-Funktion angewandt wird. „millis()“ liefert die Anzahl der Millisekunden seit dem Start (RESET).	

Schaltung



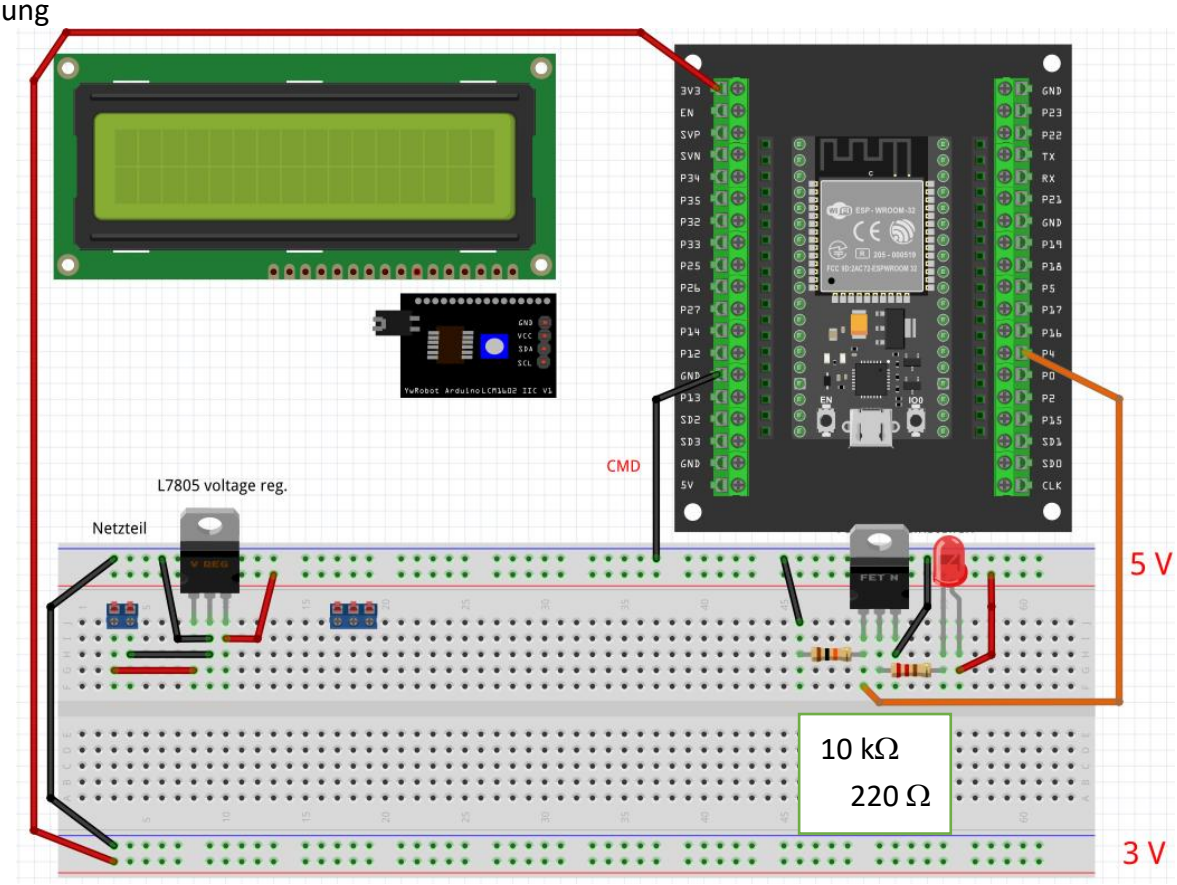
Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, Blink_No_Blocking.ino

```
const byte ledPin      = 2;       // GPIO2 (no LED on board)
unsigned long lastTime = 0;       // Variable holds the previous millis() value
const long interval    = 1000;    // Blink interval
bool ledState          = false;   // variable for toggling LED

void setup() {
  Serial.begin(115200);
  while(!Serial) {}
  Serial.println("EBW-Workshop");

  // initialize ledPin as an output.
  pinMode(ledPin, OUTPUT);
}

void loop() {
  unsigned long currentTime = millis();     // time since RESET
  if (currentTime - lastTime >= interval) { // wait a second
    ledState = !ledState;          // toggle the LED-state from high or low ...
    digitalWrite(ledPin, ledState);         // turn led on or off
    Serial.println(currentTime);
    lastTime = millis();                    // notice current time
  }
}
```

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

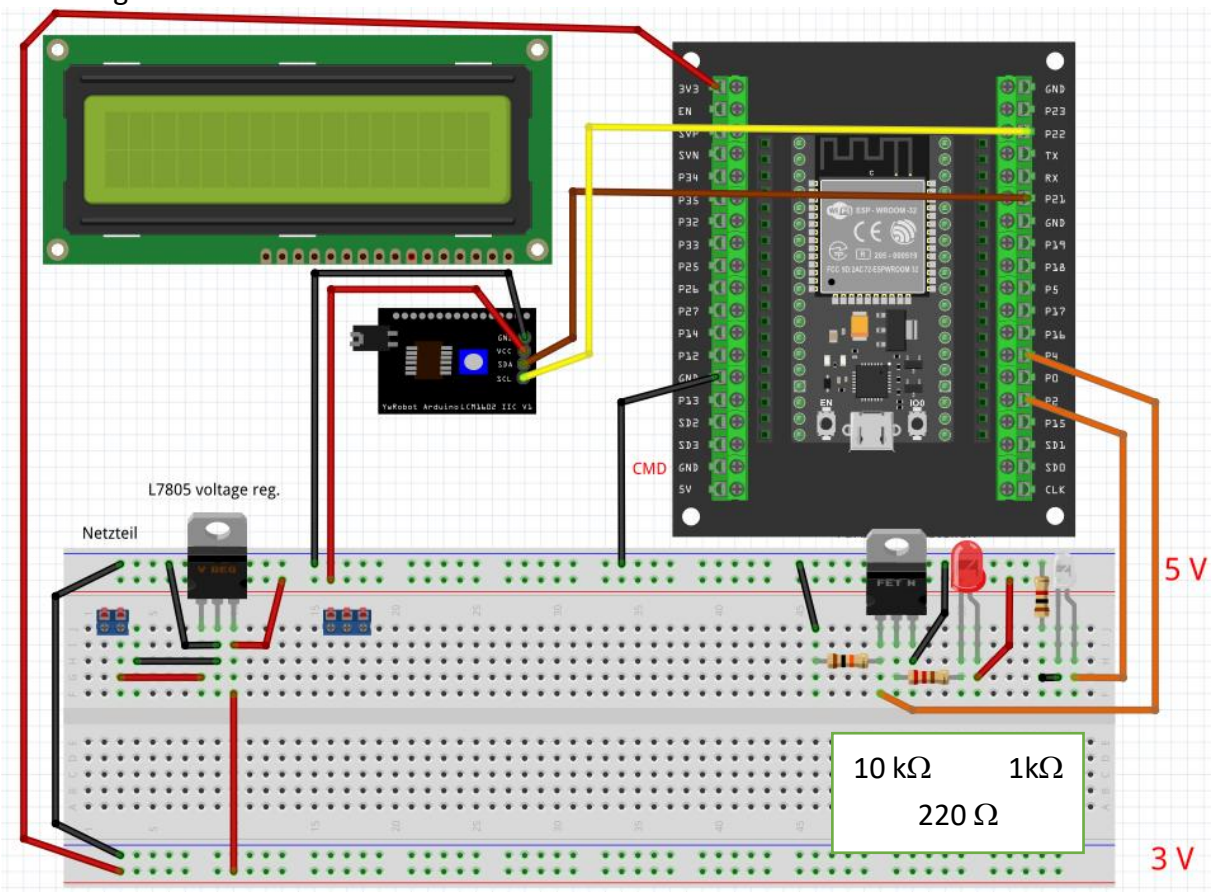
15.0 Aufgabe	LED (20 mA) steuern. Die LED soll über einen MOSFET gesteuert werden und blinken.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin Steckbrett **L7805** Spannungsregler Schraubklemme 2-polig	LED (V_f=2 V I_f=20 mA) 220 Ω Widerstand **MOSFET IRF 840b** (n-Kanal) 10 kΩ Widerstand Jumper-Kabel
Hinweise	Als Transformator kommt ein Schaltnetzteil mit max. 9 V infrage. Der Spannungsregler sorgt für 5 V Versorgungsspannung. Der ESP32 dient nicht zur Spannungsversorgung. Laut ChatGPT werden 10 mA für eine längere Lebensdauer empfohlen.	
Empfehlung	LEDs nicht direkt am GPIO betreiben (Ausnahme Low Current LED).	

Schaltung



Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, Blink_Mosfet.ino

```
#define ledPin 4              // GPIO4 where ledPin is connected to

void setup() {
  Serial.begin(115200);
  while(!Serial) {}
  Serial.println("EBW-Workshop");
  // initialize digital "ledPin" as an output.
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                 // wait for a second
  digitalWrite(ledPin, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                 // wait for a second
}
```

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
16.0 Aufgabe	LCD 16x2 einbinden. Ausgaben über LCD.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin Steckbrett Jumper-Kabel	LCD 15x2 L7805 Spannungsregler Schraubklemme 2-polig
Hinweise Empfehlung	Der ESP32 dient nicht zur Spannungsversorgung. LCD nur über externe 5 V Stromversorgung betreiben. Unbedingt Polung beachten!	
Schaltung		
Sketch siehe Folgeseite.		

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

16.1 Code Serielle Eingaben über den Monitor.

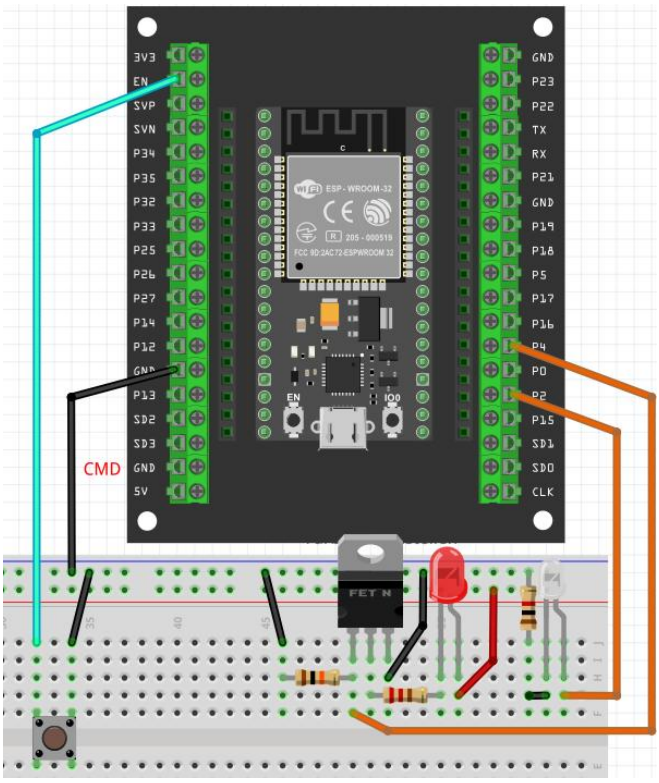
Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, LCD.ino

```
// EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“
// Sketch LCD.ino
// setting LCD 16x2 display

// Include library which needs to be installed
// LiquidCrystal I2C Frank de Brabander
#include <LiquidCrystal_I2C.h>
#define lcdColumns 16
#define lcdRows     2

// create object "lcd()" from class LiquidCrystal_I2C
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

void setup() {
  Serial.begin(115200);
  while(!Serial) {}
  Serial.println("EBW-Workshop");
  // initialize LCD
  lcd.init();
  lcd.backlight();
  lcd.print("EBW-Workshop");
}

void loop() {
  static byte sec=0;     // variable persists between calls
  delay(1000);

  // LCD-Display
  lcd.clear();
  lcd.setCursor(0, 0);  lcd.print("Seconds:");
  lcd.setCursor(9, 1);  lcd.print(sec++);
  if (sec >= 60) sec = 0;
}
```

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

17.0 Aufgabe	Ein Taster soll einen Hardware-RESET auslösen.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin Steckbrett Jumper-Kabel	L7805 Spannungsregler Schraubklemme 2-polig LCD 16x2 Taster
Problem Abhilfe	Das ESP32-Board kommt mit einem mit „RST“ bezeichneten Taster. Dieser Taster ist nicht gut zu erreichen bzw. zu betätigen. Wird der EN-Pin auf GND gezogen, wird ein Hardware-RESET ausgelöst. Ein Taster soll die kurzeitige Verbindung zu GND herstellen.	
Hinweis	Der ESP32 sendet seinen Bootstatus an die serielle Schnittstelle. Dies kann auf einfache Weise nicht verhindert werden. Erst dann kommen gewünschte serielle Ausgaben.	

Schaltung



Serieller Monitor Boot-Status

```
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:4980
load:0x40078000,len:16612
load:0x40080400,len:3480
entry 0x400805b4
```

Sketch

```
EBW
```

Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, Reset_Taster.ino

```
void setup() {
  Serial.begin(115200);
  while(!Serial) {}
  Serial.println("EBW");
}
// the loop function runs over and over again forever
void loop() {}
```

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
18.0 Aufgabe	Serielle Eingaben über den „Seriellen Monitor“. Über den „Seriellen Monitor“ sollen Informationen eingegeben werden. Das Eingabemuster soll sein: „Name:Wert“. Zum Beispiel: „LED1:1“	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin	
Aufgabe ESP32 → PC	Beim Programmieren will man wissen, wie der Sketch funktioniert. Mit Hilfe des seriellen Monitors können wir Daten vom ESP32 an den PC senden. Diese Technik wird als debugging bezeichnet. Weiterhin lassen sich Nachrichten und Werte z.B. von Sensoren ausgeben.	
PC → ESP32	Manchmal möchte man Kommandos oder Daten vom PC an den ESP32 schicken, um Parameter zu verändern.	
Serial?	Diese Art der Kommunikation setzt eine USB-Kabelverbindung zwischen PC und ESP32 voraus.	
Wichtig	Im seriellen Monitor muss „Neue Zeile“ („\n“) ausgewählt sein. Dieses „**nicht druckbare Zeichen**“ kennzeichnet das Ende einer Eingabe. Es wird im Sketch erwartet, um das Ende der Eingabe zu erkennen.	
Zum Code		
serialEvent()	Diese Funktion wird in der ESP32-Core-Library definiert (bei manchen ESP32 Boards allerdings nicht). Diese Funktion wird stets am Ende von „loop()“ ausgeführt und ist daher eigentlich keine Interrupt-Routine (Event). Alle ankommenden Zeichen werden in einem „Puffer“ abgelegt.	
	`readStringUntil()`	
	Diese Funktion hat eine blockierende Eigenschaft. So lange kein Zeilen-Endezeichen (Neue Zeile oder „\n“) erkannt wird, bleibt die Ausführung stehen.	
	`strCommand = Serial.readStringUntil(':');`	
	Alle Zeichen bis zum “:” werden als Zeichenkette (String) gelesen. Die Zeichenkette wird der Variablen „strCommand“ zugewiesen. Der Puffer wird hinter dem „:“ abgeschnitten.	
	`controlValue = Serial.readStringUntil('\n').toInt();`	
	Alle Zeichen bis zum “\n” werden als Zeichenkette (String) gelesen. Die Funktion „toInt()“ wandelt die Zeichen in einen Ganzzahlwert um. Der Ganzzahlwert wird der Variablen „`controlValue`“ zugewiesen. Der Puffer wird gelöscht.	
	`serialComplete = true;`	
	Wir teilen der loop()-Funktion mit, dass Daten über den seriellen Monitor eingegangen sind und reagieren darauf.	
Sketch siehe nächste Seite.		

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

18.1 Code Serielle Eingaben über den Monitor.

Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, Serial_Print_Read.ino

```
// EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“
// Sketch Serial_Print_Read.ino
// Get data from Serial Monitor

// The syntax of the message in Serial Monitor is: "name:value"
// E.g. "LED:1"

String strCommand;              // the received name
int controlValue;               // the received value
bool serialComplete = false;    // flag for loop() if serial input is complete

void setup() {
  Serial.begin(115200);
  while(!Serial) {}             // wait for established Serial connection
  Serial.println("EBW-Workshop");

  // serialEventRun();    // uncomment if serialEvent() does not work
  // serialEvent();       // umcomment this if serialEvent still does not work
}

void loop() {
  if (serialComplete) {     // serialComplete is set by function serialEvent()
    Serial.println( "Echo " + strCommand + ":" + controlValue);
    serialComplete = false;      // reset the flag and wait for new input
  }
}

void serialEvent() {
  byte buffer = Serial.available();
  Serial.println(buffer);        // only buffer=0 leads to false otherwise true
  if (buffer) {
    // Receiving input from Serial Monitor, Syntax LED:1 or LED:0
    strCommand    = Serial.readStringUntil(':');        // delemitter ':'
    controlValue  = Serial.readStringUntil('\n').toInt();
  }
  serialComplete = true;            // setting flag serialComplete for loop()
}
```

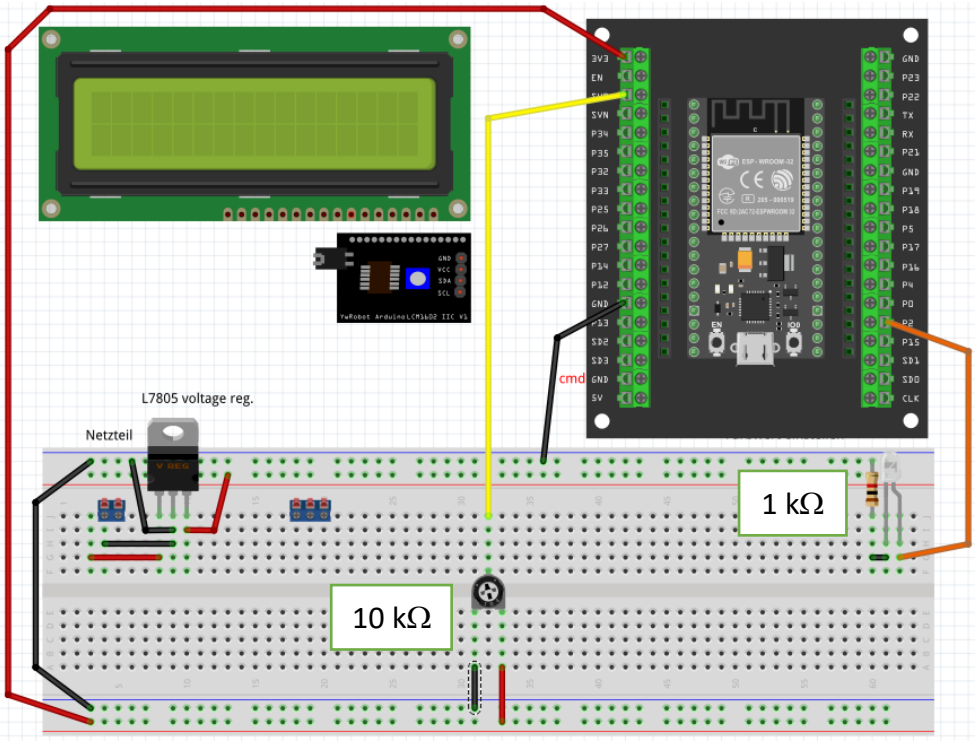
EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
19.0 Aufgabe	Serielle Eingaben über den „Seriellen Monitor“ und Timeout. Über den „Seriellen Monitor“ werden Informationen eingegeben. Das Einlesen der Daten kann, je nach Code, ein blockierendes Verhalten zeigen. Dies Verhalten soll untersucht werden.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin	
Problem	`readStringUntil()`	
Ursache	Diese Funktion hat eine blockierende Eigenschaft. So lange kein Zeilen-Endezeichen (Neue Zeile oder „\n“) erkannt wird, bleibt die Ausführung stehen.	
Beispiel	Lautet die Funktion z.B. „`readStringUntil('\n')`“, so muss die Eingabe mit „Neue Zeile“ abschließen. Das ist in den Einstellungen des „Seriellen Monitors“ einzustellen.	
Timeout	In oben genannten Fall wird die Ausführung nach einem „Timeout“ von 1 Sekunde fortgesetzt.	
Idee	Das Verhalten sichtbar machen, in dem eine eine blinkende LED beobachtet wird. Der Code der blinkenden LED muss nicht blockieren sein.	
Test	Was beobachten wir bei den Eingaben von: • LED:1 • LED • Leereingabe	
Ergebnis	Fehleingaben lösen einen Timeout aus.	
Sketch siehe nächste Seite.		

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

19.1 Code Serielle Eingaben über den „Seriellen Monitor“ und Timeout.

Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, Serial_Print_Read_LED.ino

```
String strCommand;                // the received name
int controlValue;                 // the received value
bool serialComplete = false;      // flag for loop() if serial input is complete

const byte ledPin      =   2;     // GPIO of low current LED
unsigned long lastTime =   0;     // variable holds the previous millis() value
const long interval    = 100;     // Blink interval in milliseconds
bool ledState          = false;   // used for toggling LED

void setup() {
  Serial.begin(115200);
  while(!Serial) {}               // wait for established Serial connection
  Serial.println("EBW-Workshop");
  pinMode(ledPin, OUTPUT);

  // serialEventRun();      // uncomment if serialEvent() does not work
  // serialEvent();         // umcomment this if serialEvent still does not work
}

void loop() {
  if (serialComplete) {     // checking for serial data: true if data received
    Serial.println( "Echo " + strCommand + ":" + controlValue);
    serialComplete = false;      // reset the flag and wait for new input
  }
  blinkLED();                    // call function for blinking LED
}

// function (depending on core) which runs if seriel data are present
void serialEvent() {
  if (Serial.available()) {
    // Receiving input from Serial Monitor, Syntax LED:1 or LED:0
    // read a string up to character ':'
    strCommand    = Serial.readStringUntil(':');
    // read the remaining input and convert it to int
    controlValue  = Serial.readStringUntil('\n').toInt();
  }
  serialComplete = true;        // setting flag serialComplete for loop()
}

void blinkLED() {
  // get the currenttime counting since RESET
  unsigned long currentTime = millis();
  if (currentTime - lastTime >= interval) {    // wait for an interval
    ledState = !ledState;                      // toggle the LED-state
    digitalWrite(ledPin, ledState);            // turn led on or off
    lastTime = millis();                       // notice current time
  }
}
```

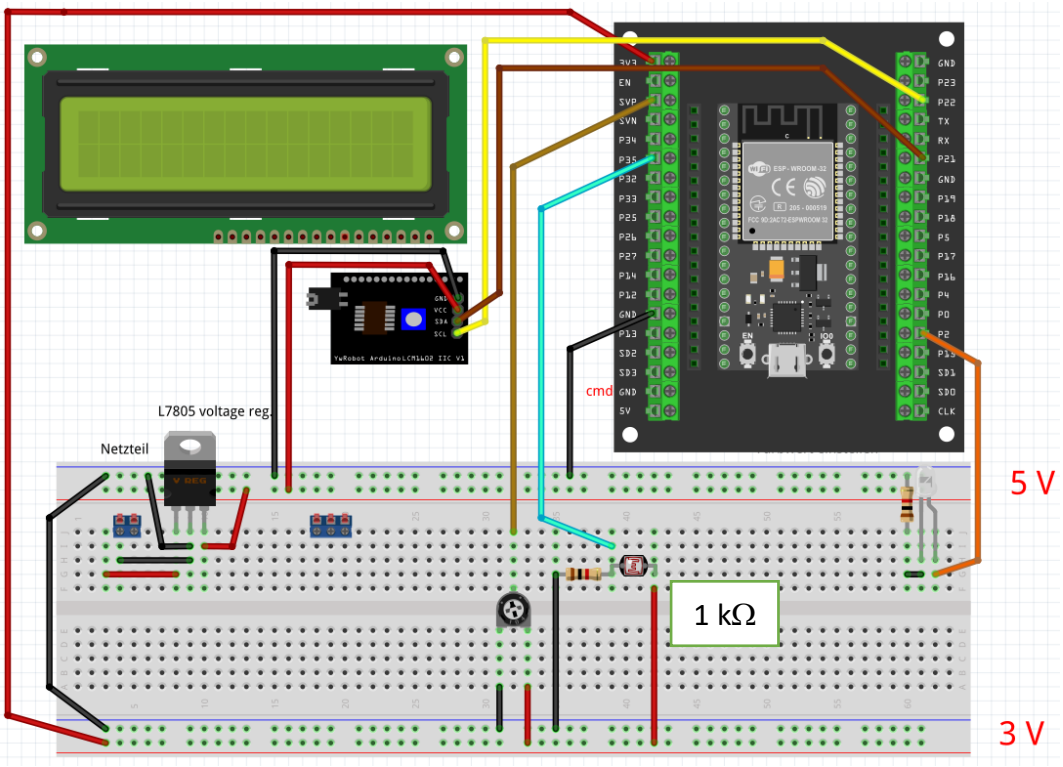
EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
20.0 Aufgabe	Mit einem Potentiometer eine LED dimmen.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin Steckbrett Jumper-Kabel	LED Low Current (V_F = **1,8 V** I_F = **2 mA**) 1 kΩ Widerstand 10 kΩ Trimmer
Achtung	Die GPIOs sollten prinzipiell nicht mit Spannungen größer 3,3 V belastet werden. Höhere Spannungen führen zur Zerstörung.	
Schaltung		
Hinweise	Das Auslesen eines Analogwerts mit dem ESP32 ermöglicht die Messung unterschiedlicher Spannungspegel zwischen 0 V und 3,3 V. Die gemessene Spannung wird anschließend einem Wert zwischen 0 und 4095 zugeordnet, wobei 0 V dem Wert 0 und 3,3 V dem Wert 4095 entspricht. Das Verhältnis zwischen Spannung und ADC-Wert ist nicht linear.	
ADC GPIOs	Vorhanden sind Kanäle als ADC1 und ADC2 bezeichnet. Hier verwende ich ADC1-0 entsprechend GPIO36 (Bord-Bezeichnung SP).	
Achtung	Ändere nicht die „attenuation“ (Dämpfung). Der Standardwert ist „ADC_11db“, d.h. geeignet für 0 bis 3,3 V. Anweisung: `analogSetAttenuation(attenuation);`	
Zum Code	`analogRead(potiPin);`	
	Liest den analogen Wert und gibt ihn in einen Wertebereich von 0 bis 4095 zurück.	
	`map(analogValue, 0, 4095, 0, 255);`	
	Funktion, die den gelesenen Analogwert in einem PWM-Wert von 0 bis 255 umrechnet.	
Sketch siehe nächste Seite.		

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

20.1 Code Mit einem Potentiometer eine LED dimmen.

Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, LED_ADC.ino

```
// EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“
// Sketch LED_PWM.ino
// Attention: Never supply potentiometer with a volatage higher than 3.3 V
//            Never change the attenuation: default "ADC_11db"
// the ADC has a 12 bit resolution from 0 to 4095 (2^12-1)

#define potiPin  36  // ESP32 pin GPIO36 (ADC0) connected to Potentiometer pin
#define ledPin    2  // ESP32 pin GPIO2 connected to LED's pin

void setup() {
  Serial.begin(115200);
  while(!Serial) {}
  Serial.println("EBW-Workshop");

  // Initialize ledPin as an output
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // reads the input from analog pin A0 (value between 0 and 4095)
  short analogValue = analogRead(potiPin);

  // the analogRead() values for 12 bit resolution, range between 0 to 4095
  // scale poti-values to led-brightness: PWM value between 0 and 255
  short brightness = map(analogValue, 0, 4095, 0, 255);

  // sets the brightness LED that connects to ledPin
  analogWrite(ledPin, brightness);

  // print out the value
  Serial.print("Analog value = ");
  Serial.print(analogValue);
  Serial.print(" => brightness = ");
  Serial.println(brightness);
  delay(500);
}
```

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
21.0 Aufgabe	Mit einem Fotowiderstand das Umgebungslicht messen. Einen ADC-Eingang verwenden. Ausgaben über LCD.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin Steckbrett Jumper-Kabel	LCD 15x2 L7805 Spannungsregler Schraubklemme 2-polig Fotowiderstand
Empfehlung	LCD nur über externe 5 V Stromversorgung betreiben.	
Achtung	Die GPIOs prinzipiell nicht mit Spannungen größer 3,3 V belasten.	
	Ändere nicht die „attenuation“ (Dämpfung). Der Standardwert ist „ADC_11db“, d.h. geeignet für 0 bis 3,3 V. Anweisung: `analogSetAttenuation(attenuation);`	
ADC GPIOs	Vorhanden sind Kanäle als ADC1 und ADC2 bezeichnet. Hier verwende ich ADC1-7 entsprechend GPIO35.	
Fotowiderstand LDR	Der Widerstandswert ändert sich abhängig von der Intensität des Lichts. Bei Helligkeit verringert sich der Widerstand des Fotowiderstands. Bei Dunkelheit steigt der Widerstand.	
Zum Code	`analogRead(ldrPin);`	
	Liest den analogen Wert und gibt ihn in einen Wertebereich von 0 bis 4095 zurück.	
	`map(analogValue, 0, 4095, 0, 100);`	
	Funktion, die hier den gelesenen Analogwert in einen Prozentwert von 0 bis 100 umrechnet.	
Widerstand	In Abhängigkeit vom Widerstand des LDR habe wurde ein Widerstand von 1 kΩ gewählt	
Schaltbild		
Sketch siehe nächste Seite.		

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

21.1 Code Mit einem Fotowiderstand das Umgebungslicht messen.

Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, LDR_ADC.ino

```
// EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“
// Sketch LDR_ADC.ino
// Attention: Never supply potentiometer with a volatage higher than 3.3 V
//            Never change the attenuation: default "ADC_11db"
// the ADC has a 12 bit resolution from 0 to 4095 (2^12-1)

// setting LCD 16x2 display
#include <LiquidCrystal_I2C.h>
#define lcdColumns 16
#define lcdRows     2
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

#define ldrPin 35  // ESP32 pin GPIO35 (ADC1-7) connected to LDR

void setup() {
  Serial.begin(115200);
  while(!Serial) {}
  Serial.println("EBW-Workshop");

  // initialize LCD
  lcd.init();
  lcd.backlight();
  lcd.print("EBW-Workshop");
}
```

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“		
22.0 Aufgabe	Classic Bluetooth erste Schritte. Nachrichten zwischen „Seriellen Monitor“ und Smarphone austauschen.	
Bauteile	ESP32 Dev Kit C V2 ESP32 Terminal Adapter 38 Pin	
Achtung	Diese Aufgabe verlangt ein Android Smartphone, da iOS Classic-Bluetooth nicht unterstützt. (Huawei geht nicht, weil kein Zugriff auf „Google Play Store“.	
Smartphone vorbereiten	Im Google Play Store nach der App „“Serial Bluetooth terminal” suchen und installieren.	
Bluetooth Classic (ChatGPT)	Bluetooth Classic ist eine Funktechnologie für kurze Distanzen (typisch 1–10 m), die eine **dauerhafte Verbindung** zwischen zwei Geräten aufbaut und **relativ hohe Datenraten** erlaubt (bis zu 3 Mbit/s). Es handelt sich um die „alte“ Bluetooth-Variante – im Gegensatz zu **Bluetooth Low Energy (BLE)**, das später hinzugekommen ist.	
Erkennung	Geräte suchen einander und tauschen grundlegende Informationen wie Name und Klasse aus.	
Koppeln	Austausch eines PIN-Codes oder Sicherheitsschlüssels. Beide speichern die Verbindung für spätere Nutzung.	
Datenaustausch	Danach wird kontinuierlich über einen Datenkanal kommuniziert.	
Verbindung herstellen	Bluetooth auf ESP32 starten. Auf Smartphone in Einstellungen, Gerät „ESP32-DeinName” koppeln. App „Serial Bluetooth Terminal” Device einrichten. Symbol „Verbinden” drücken. Nachrichten senden/empfangen	
Besonderheit	Die hier verwendete Library hat keine Pin-Funktion zur Zugangskontrolle!	
Zum Code	`#include "BluetoothSerial.h"`	
	ESP32 Bluetooth-Library einbinden.	
	`BluetoothSerial SerialBT`	
	Das Objekt „SerialBT“ der Klasse BluetoothSerial erzeugen.	
	`SerialBT.begin("EBW-DeinName")`	
	Bitte einen eigenen „Device Name“ eingeben. Initialisieren von Bluetooth auf dem ESP32.	
	`SerialBT.available()`	
	Besteht eine Bluetooth-Kopplung mit einem Smartphone, dann Daten empfangen oder senden.	
	`SerialBT.write(…)`	
	Vom z. B. „Serial Monitor“ Daten an das Smartphone senden.	
	`SerialBT.read()`	
	Vom Smartphone eingegangene Daten lesen.	
Quellen	https://randomnerdtutorials.com/esp32-bluetooth-classic-arduino-ide/	
Sketch siehe nächste Seite.		

EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“

22.1 Code Classic Bluetooth erste Schritte.

Sketch: „Github.com/EKlatt/Experiences“, Ordner ESP32, Bluetooth_Classic.ino

```
// EBW: Experimentieren mit dem „ESP-32 Dev Kit C V2“
// Sketch Bluetooth_Classic.ino
// Get a message from Serial Monitor and send it to Bluetooth-Device
// Get a message from  Bluetooth-device and send it to Serial Monitor

// https://randomnerdtutorials.com/esp32-bluetooth-classic-arduino-ide/
// https://lastminuteengineers.com/esp32-bluetooth-classic-tutorial/

#include "BluetoothSerial.h"        // Library einfügen

BluetoothSerial SerialBT;            // create the object SerialBT from class
BluetoothSerial

void setup() {
  // Begin serial communication with ESP32 and Arduino IDE (Serial Monitor)
  Serial.begin(115200);
  while(!Serial) {}                  // wait for established serial connection
  Serial.println("EBW-Workshop");

  // Initialize the Bluetooth stack
  // The Bluetooth device name appears on your smartphone, please change it
  if (!SerialBT.begin("EBW-Enno")) {   // Bluetooth device name
    Serial.println("Bluetooth failed to intit!");
    while(1);
  }
  Serial.println("Bluetooth bereit. Jetzt mit Smartphone koppeln.");
}

void loop() {
  if (Serial.available()) {           // if there is an input from Serial
Monitor (PC)
    SerialBT.write(Serial.read());    // send the input to the Bluetooth-Device
  }
  if (SerialBT.available()) {         // if there is an input from Bluetooth-
Device
    Serial.write(SerialBT.read());    // send the input to Serial Monitor (PC)
  }
  delay(20);
}
```

Installation der Arduino IDE 2.3.6

Einführung in die Arduino IDE 2.3.6

Einrichten des Boards ESP32

LED mit Potentiometer dimmen

Schrittmotor mit Drehregler steuern

LDR-Signale und Serieller Plotter

LM35-Signale und Serieller Plotter

DHT11-Signale und Serieller Plotter

Schrittmotor mit Seriellem Monitor steuern

Einführung in Classic Bluetooth (funktioniert nur mit Android Smartphones)

NeoPixel-LED-Streifen steuern

…