FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

# Microcontroller Fundamentals

UNIVERSITY OF APPLIED SCIENCES – FH CAMPUS VIENNA

FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

# Chapter overview

> Microcontroller
> ESP32 – WROOM Kit
> Arduino IDE
> Serial Communication

# Microcontroller

A microcontroller contains a microprocessor, which is defined by its instruction set, i.e. its machine instructions (assembly language).

Different assembler commands are usually required to program an Xtensa, RISC-V or ARM Cortex-M CPU.

Programs for microcontrollers are usually written in the C programming language.

The "Integrated Development Environment" (IDE) includes a C compiler that converts the programs into machine language. Therefore, programs can be "simply" translated (compiled) for the corresponding processor architecture. Each processor has a "program counter", "stack pointer" and several registers.

During "flashing" the program is transferred from the development PC to the non-volatile program memory. Flash memory is a further development of EEPROM technology.

# Microcontroller -> Embedded Systems

The microcontroller is embedded in a system, with the μC itself being soldered to a control board.

Universal boards that are used for product development, testing of individual functions, or exploratory learning are referred to as evaluation boards or development kits.

These circuit boards usually have one or more LEDs and buttons which are connected to individual connections of the microcontroller (pins) via the printed circuit board conductors to form electronic circuits.

The pins are commonly referred to as GPIO "General Purpose Input Output" and can contain a binary 1, represented by an electrical voltage of +3.3V (older μC may use +5V), and a binary 0, represented by 0V (also known as Ground GND).

# How to program a microcontroller

Using the Arduino IDE, the configuration takes place on a "higher abstracted" level, but this means that not all functions can be configured.

The solutions are reached quickly but are not optimal
(energy consumption & computing power)

With more detailed knowledge, significantly better solutions can be achieved. The path to these solutions requires familiarization with the details, which can be found in the documentation, as well as becoming familiar with the tools and library functions provided by the manufacturers. ESP32 microcontrollers are programmed using Microsoft Visual Code in combination with platform.io or the Toolchain provided by espressif!

# Include derictive & Header files

Many programming languages and other computer files have a directive, often called **include** (sometimes copy or import), that causes the contents of the specified file to be inserted into the original file.
These included files are called copybooks or **header files**. There are over one thousand C library files and they are often used to define the physical layout of program data, pieces of procedural code, and/or forward declarations while promoting encapsulation and the reuse of code or data.

In computer programming, a header file is a file that allows programmers to separate certain elements of a program's source code into reusable files. Header files commonly contain forward declarations of classes, subroutines, variables, and other identifiers. Programmers who wish to declare standardized identifiers in more than one source file can place such identifiers in a single header file, which other code can then include whenever the header contents are required. This is to keep the interface in the header separate from the implementation.

The C standard library and the C++ standard library traditionally declare their standard functions in header files.

https://en.wikipedia.org/wiki/Include_directive

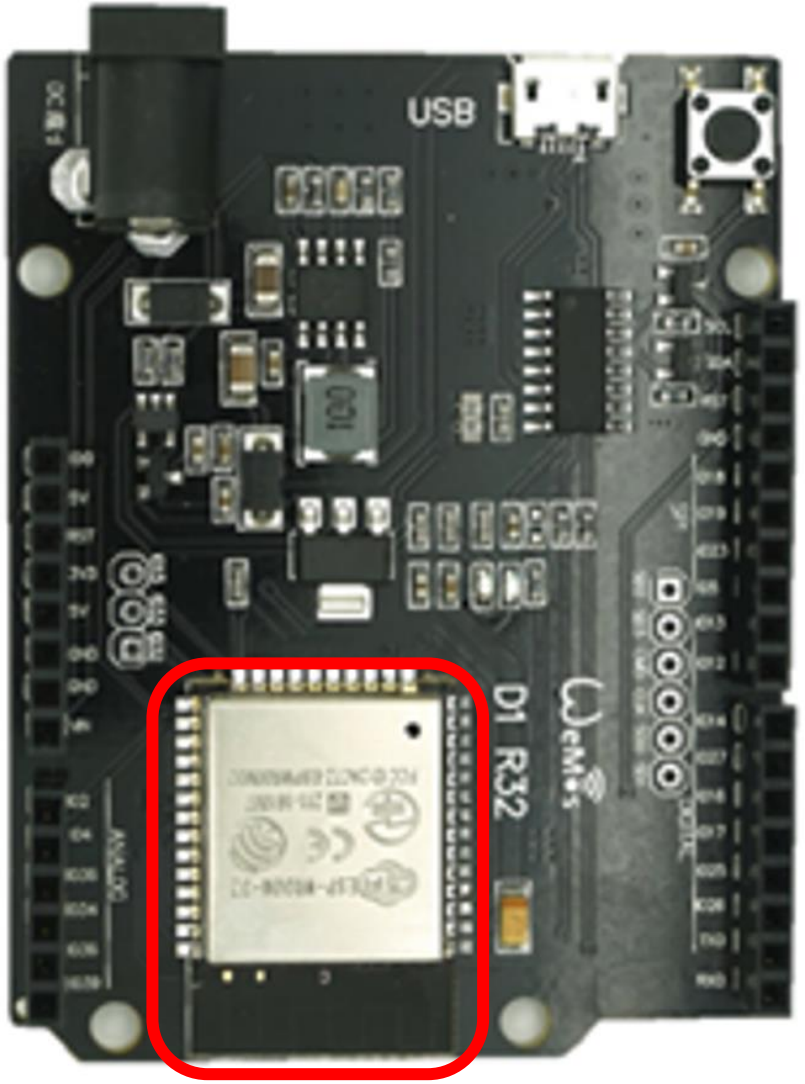
# ESP32 Wroom Board

**This development kit for the ESP32 has a USB connection and the necessary components for power supply.**

**The assignment of the pins can be seen in this figure.**

**To make an LED blink, we need to connect an LED and a resistor to this module. For this we need to use a breadboard.**

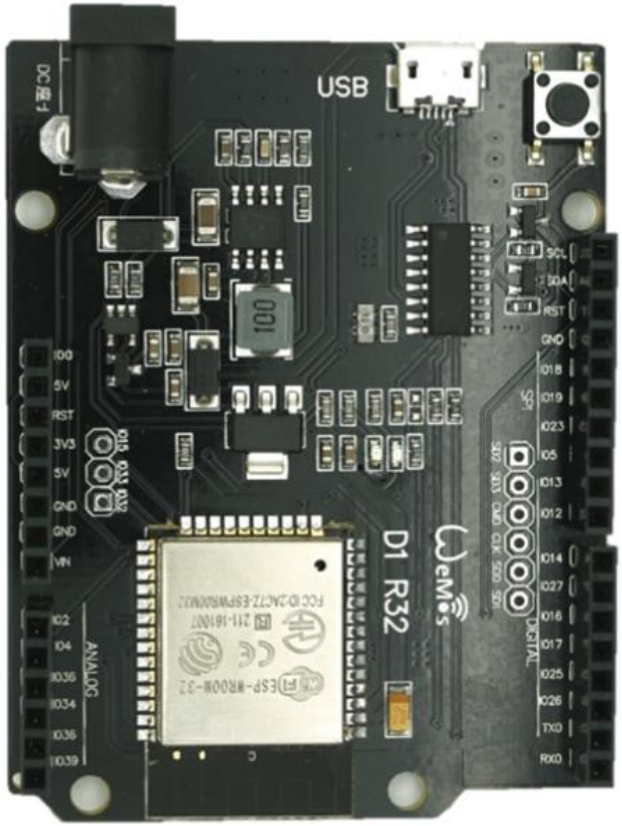
**This Evaluation Board has an ESP32-Wroom Module soldered on Top of the PCB.**




Wemos D1 R32 (based on ESP32)

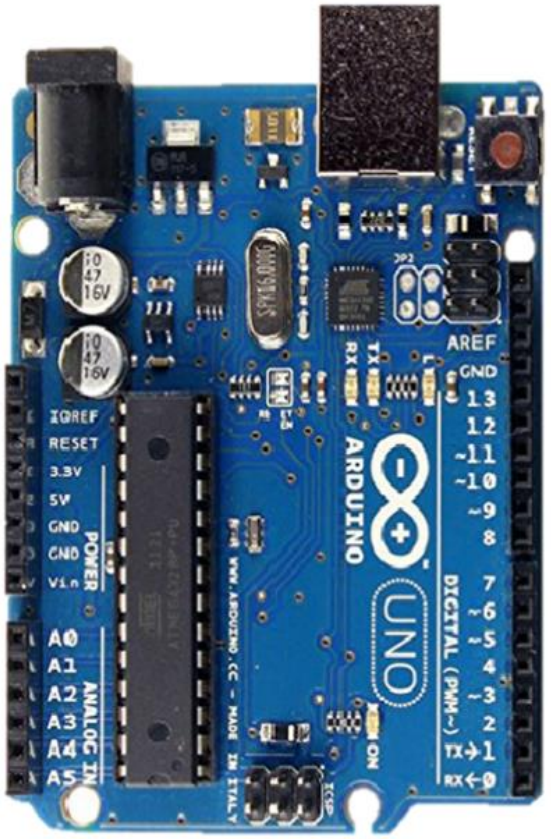
| Wemos D1 R32 |
|---|
| IO26 |
| IO25 |
| IO17 |
| IO16 |
| IO27 |
| IO14 |
| IO12 |
| IO13 |
| IO5 |
| IO23 |
| IO19 |
| IO18 |
| IO2 |
| IO4 |
| IO35 |
| IO34 |
| IO36 |
| IO39 |

# ESP32 Wroom Board




Wemos D1 R32 (based on ESP32)

| Wemos D1 R32 | Arduino Uno |
|---|---|
| IO26 | D2 |
| IO25 | D3 |
| IO17 | D4 |
| IO16 | D5 |
| IO27 | D6 |
| IO14 | D7 |
| IO12 | D8 |
| IO13 | D9 |
| IO5 | D10 |
| IO23 | D11 |
| IO19 | D12 |
| IO18 | D13 |
| IO2 | A0 |
| IO4 | A1 |
| IO35 | A2 |
| IO34 | A3 |
| IO36 | A4 |
| IO39 | A5 |




Arduino Uno (Atmega328)

**The assignment of the IO-Pins from the ESP32-Microcontroller to the Arduino-Connector can be seen in this table!**

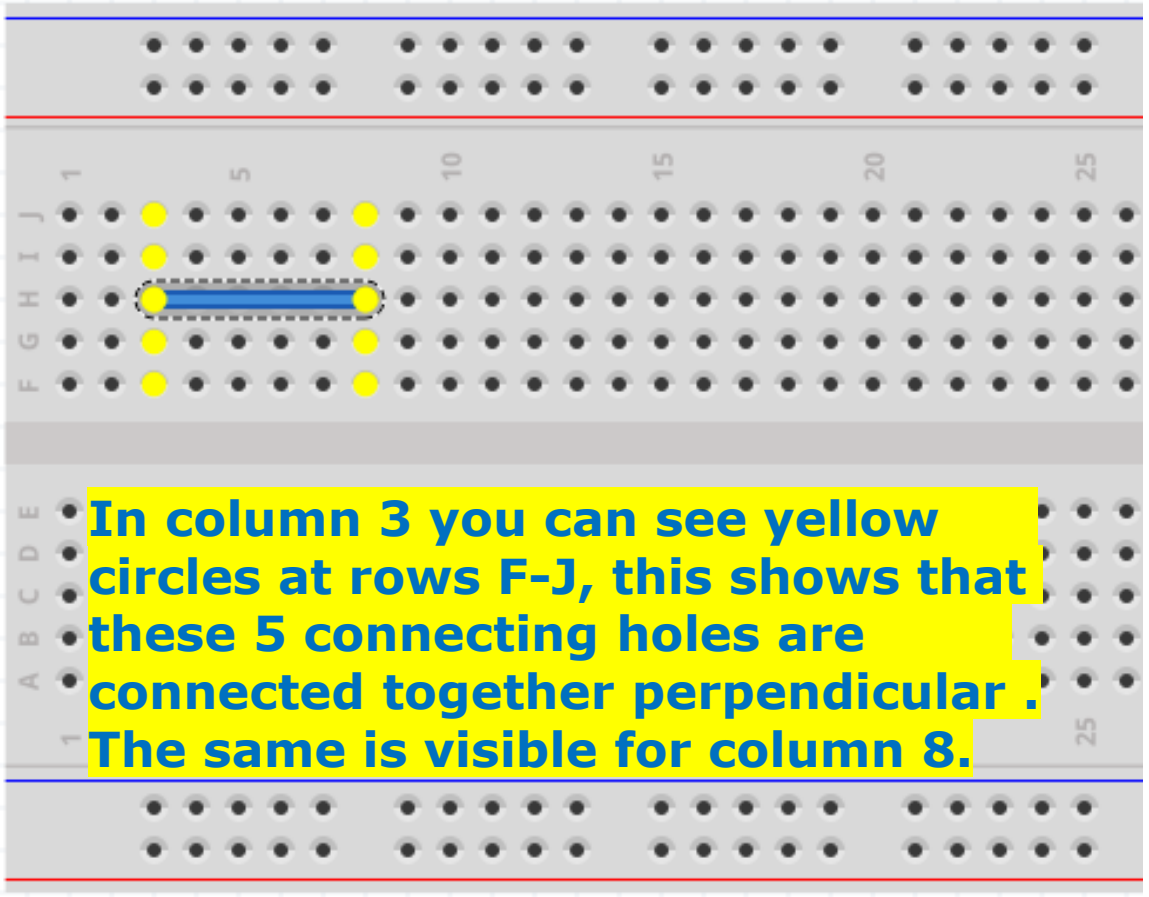
# Breadboard

A breadboard has 10 rows labeled with the letters A through J

The columns are numbered, starting with 1 on the left

In each column rows A-E and **separately** rows F-J are connected!

With the **blue cable** shown in line H, the connecting holes F-J in column 3 can be tied with the connecting holes F-J in column 8.




**In column 3 you can see yellow circles at rows F-J, this shows that these 5 connecting holes are connected together perpendicular . The same is visible for column 8.**

https://www.elektronik-kompendium.de/sites/praxis/bauteil_steckbrett.htm

# ESP32 Wroom and Arduino IDE - 1

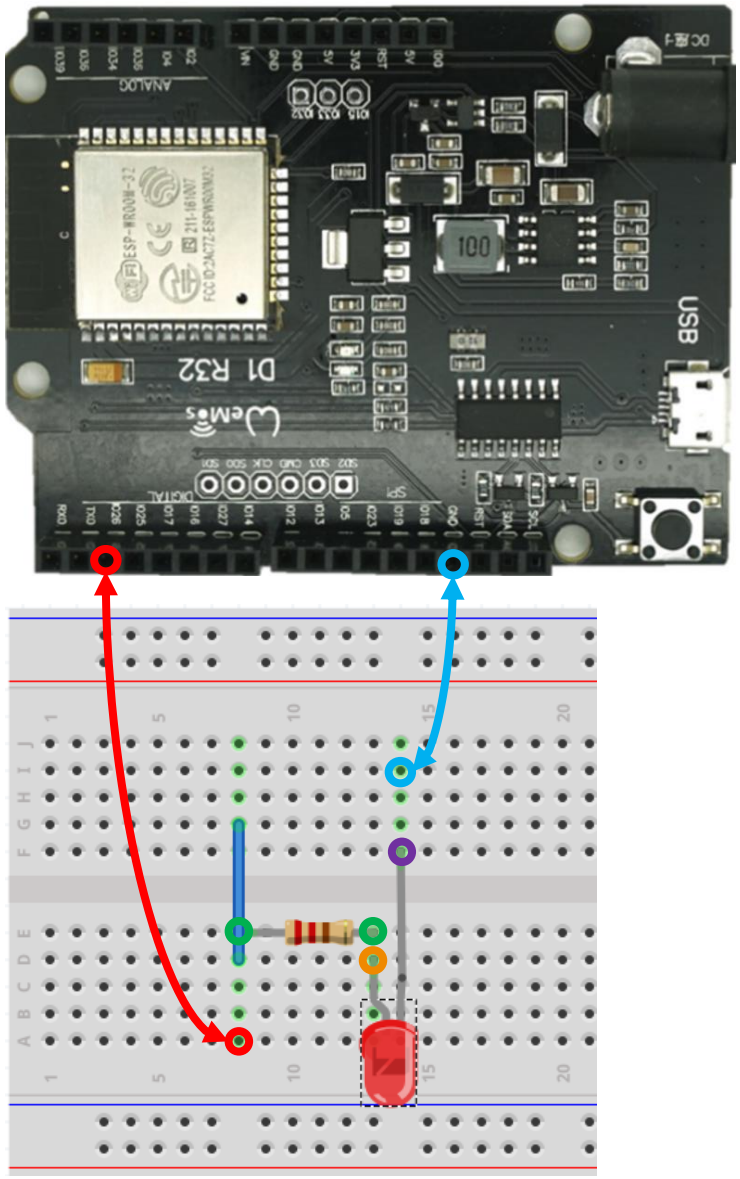
1) We start with the IO26
which can be found at Arduino D2,
with a jumpwire we connect
to the breadboard

3) The resistance is inserted in row E
plugged into columns 8 and 14

4) The LED has its plus pole on the longer
of the two connections.
The positive pole is connected to
the resistor on the breadboard,
since both are plugged into column 14.

5) The negative pole of the LED is plugged
into column 14 row F

6) With another jumpwire we make
a connection to the ESP32 ground pin (0V).



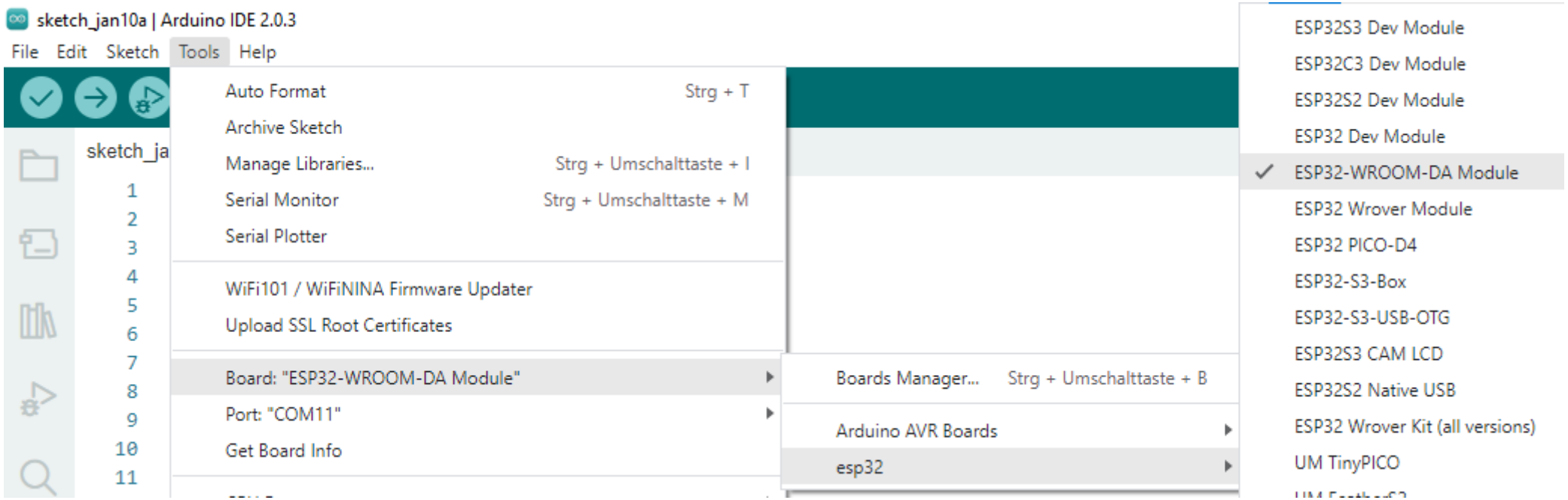
# How to program with Arduino IDE

In the Arduino IDE,
a “sketch” is used
instead of the main.c file.

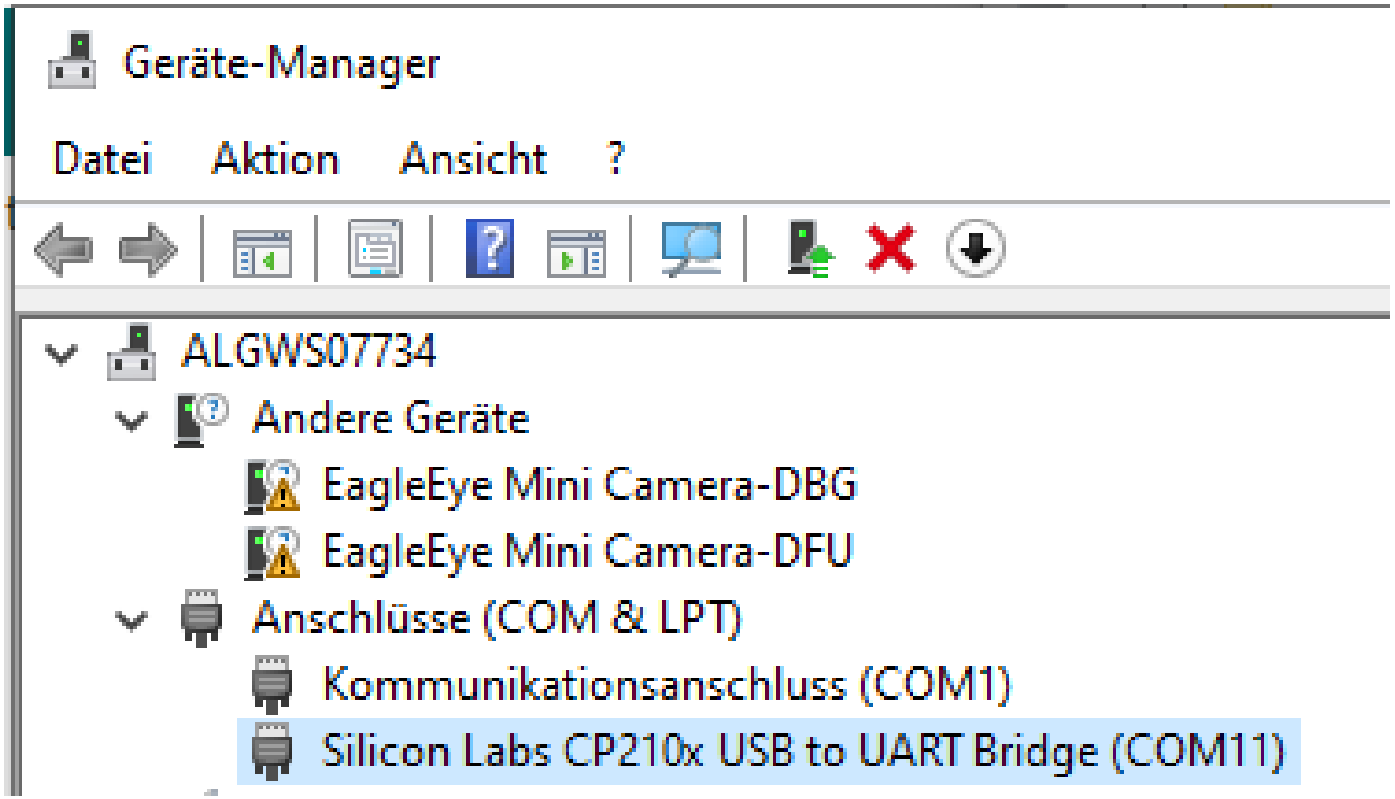
There are two functions that are empty at startup:

- All program parts that are to be executed **only once** after a reset are inserted under **setup()**.
- All program parts that are to be executed **permanently**, in an endless loop, after the setup are inserted under **loop()** .

# Arduino IDE – select ESP32 Board

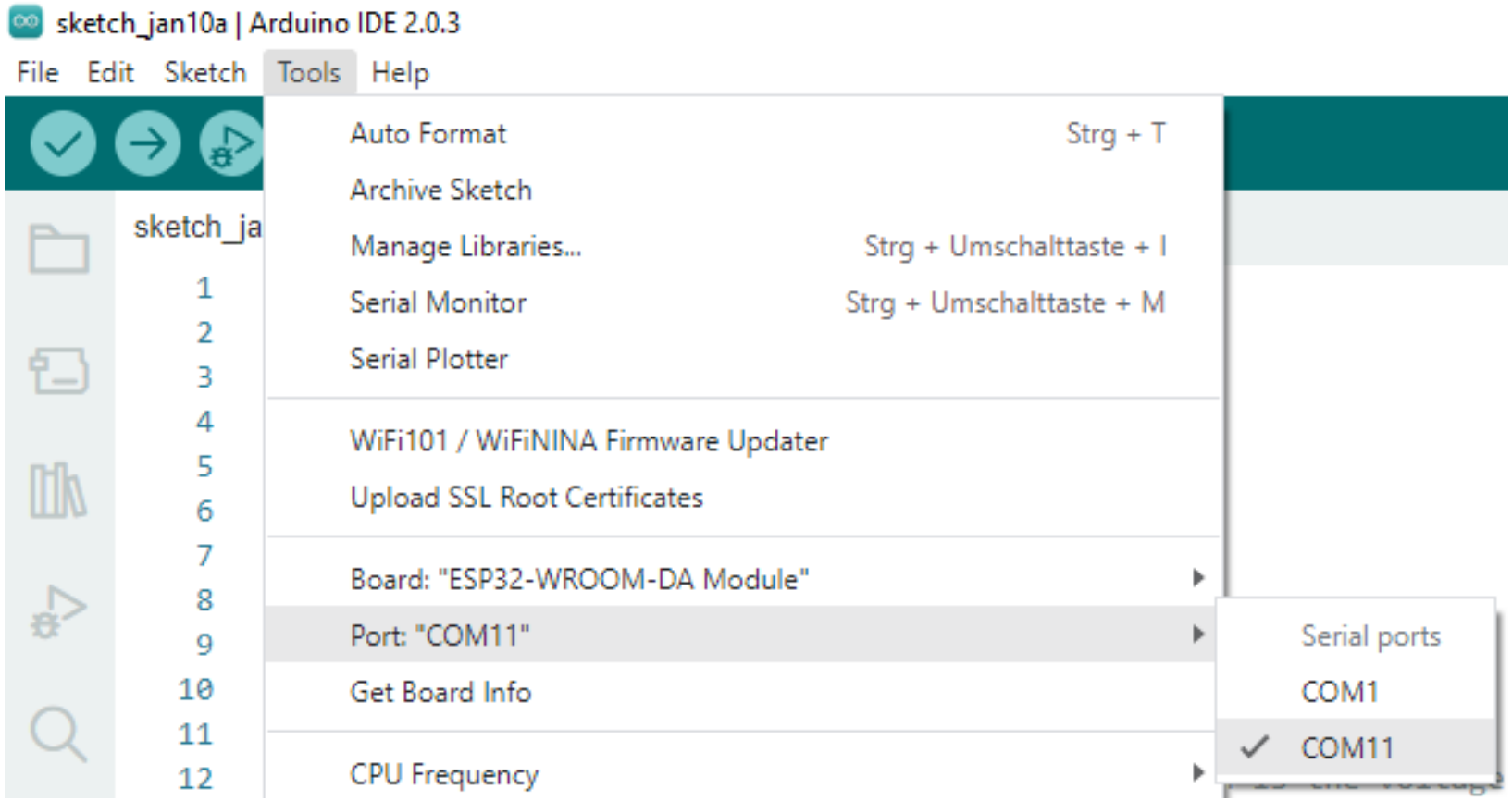


# Windows Device-Manager




**Communication between the development PC and the ESP32 takes place via a CP210x chip from Silicon Labs. The COM port assigned under Windows must be looked up in the device manager!**

# Arduino IDE – select COM Port



# Program with Arduino IDE

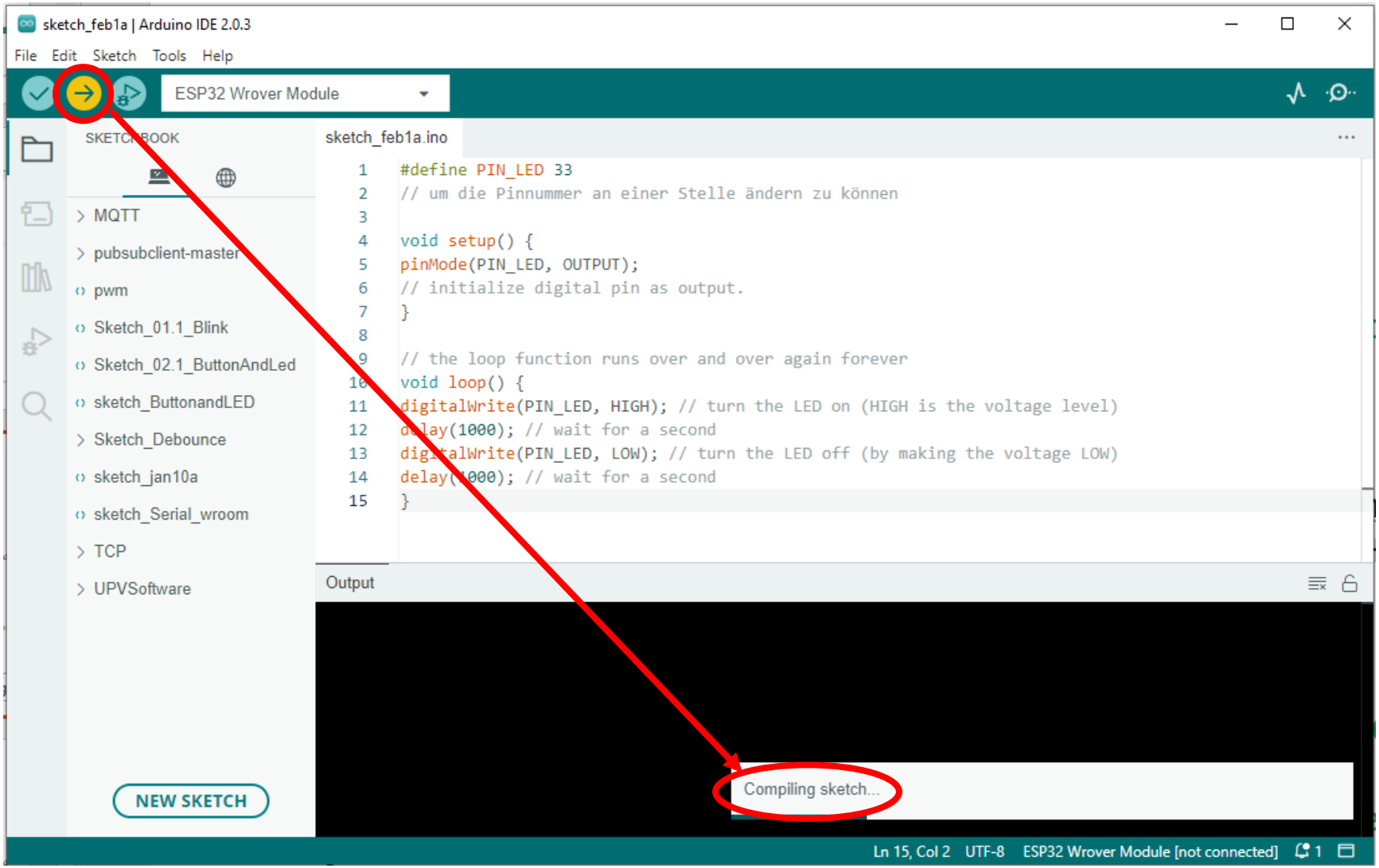
```
#define PIN_LED 26
// to modify the Pin number
// only at one place in your file

void setup() {
pinMode(PIN_LED, OUTPUT);
// initialize digital pin as output.
}

// the loop function runs over and over again forever
void loop() {
digitalWrite(PIN_LED, HIGH); // turn the LED on (HIGH is the voltage level)
delay(1000); // wait for a second

digitalWrite(PIN_LED, LOW); // turn the LED off (by making the voltage LOW)
delay(1000); // wait for a second
}
```

# How to program with Arduino IDE



# Arduino IDE

**Several steps will be done by the IDE**

**Build**

**Connect**

**Erase**

**Write**

**Reset & Run**

Output

```
Sketch uses 235049 bytes (17%) of program storage space. Maximum is 1310720 bytes.
Global variables use 21800 bytes (6%) of dynamic memory, leaving 305880 bytes for local variables. Maximum is 327680 bytes.
esptool.py v4.2.1
Serial port COM11
Connecting....
Chip is ESP32-D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, Coding Scheme None
Crystal is 40MHz
MAC: 30:ae:a4:40:a7:c8
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Configuring flash size...
Flash will be erased from 0x00001000 to 0x00005fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Flash will be erased from 0x0000e000 to 0x0000ffff...
Flash will be erased from 0x00010000 to 0x00049fff...
Compressed 18880 bytes to 13017...
Writing at 0x00001000... (100 %)
Wrote 18880 bytes (13017 compressed) at 0x00001000 in 0.4 seconds (effective 370.0 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 146...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (146 compressed) at 0x00008000 in 0.1 seconds (effective 428.7 kbit/s)...
Hash of data verified.
Compressed 8192 bytes to 47...
Writing at 0x0000e000... (100 %)
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.1 seconds (effective 642.0 kbit/s)...
Hash of data verified.
Compressed 235440 bytes to 129037...
Writing at 0x00010000... (12 %)
Writing at 0x0001e473... (25 %)
Writing at 0x000243cb... (37 %)
Writing at 0x000296ab... (50 %)
Writing at 0x0002ec68... (62 %)
Writing at 0x00037156... (75 %)
Writing at 0x0003f275... (87 %)
Writing at 0x0004482f... (100 %)
Wrote 235440 bytes (129037 compressed) at 0x00010000 in 2.3 seconds (effective 804.4 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

# Microcontroller programming using Arduino IDE

The development environment brings you to a working result very quickly, **WITHOUT** having to read the user manuals for exact details about the microcontroller used.

With **pinMode(Output)** the desired connection can be configured as a digital output.

The programmer does not need any knowledge about the microcontroller, but instead knowledge about the functions provided by the respective IDE!

**https://www.arduino.cc/reference/en/**

# Arduino Function reference

**Digital I/O**

digitalRead()
digitalWrite()
pinMode()

**Analog I/O**

analogRead()
analogReference()
analogWrite()

**Zero, Due & MKR Family**

analogReadResolution()
analogWriteResolution()

**Advanced I/O**

noTone()
pulseIn()
pulseInLong()
shiftIn()
shiftOut()
tone()

**Time**

delay()
delayMicroseconds()
micros()
millis()

**Math**

abs()
constrain()
map()
max()
min()
pow()
sq()
sqrt()

**Trigonometry**

cos()
sin()
tan()

**Characters**

isAlpha()
isAlphaNumeric()
isAscii()
isControl()
isDigit()
isGraph()
isHexadecimalDigit()
isLowerCase()
isPrintable()
isPunct()
isSpace()
isUpperCase()
isWhitespace()

**Random Numbers**

random()
randomSeed()

**Bits and Bytes**

bit()
bitClear()
bitRead()
bitSet()
bitWrite()
highByte()
lowByte()

**External Interrupts**

attachInterrupt()
detachInterrupt()

**Interrupts**

interrupts()
noInterrupts()

**Communication**

Serial
SPI
Stream
Wire

## Variables

Arduino data types and constants.

**Constants**

HIGH | LOW
INPUT | OUTPUT | INPUT_PULLUP
LED_BUILTIN
true | false
Floating Point Constants
Integer Constants

**Conversion**

(unsigned int)
(unsigned long)
byte()
char()
float()
int()
long()
word()

**Data Types**

array
bool
boolean
byte
char
double
float
int
long
short
size_t
string
String()
unsigned char
unsigned int
unsigned long
void
word

# Arduino Structure

**Sketch**

loop()
setup()

**Control Structure**

break
continue
do...while
else
for
goto
if
return
switch...case
while

**Further Syntax**

#define (define)
#include (include)
/* */ (block comment)
// (single line comment)
; (semicolon)
{} (curly braces)

**Arithmetic Operators**

% (remainder)
* (multiplication)
+ (addition)
- (subtraction)
/ (division)
= (assignment operator)

**Comparison Operators**

!= (not equal to)
< (less than)
<= (less than or equal to)
== (equal to)
> (greater than)
>= (greater than or equal to)

**Boolean Operators**

! (logical not)
&& (logical and)
|| (logical or)

**Pointer Access Operators**

& (reference operator)
* (dereference operator)

**Bitwise Operators**

& (bitwise and)
<< (bitshift left)
>> (bitshift right)
^ (bitwise xor)
| (bitwise or)
~ (bitwise not)

**Compound Operators**

%= (compound remainder)
&= (compound bitwise and)
*= (compound multiplication)
++ (increment)
+= (compound addition)
-- (decrement)
-= (compound subtraction)
/= (compound division)
^= (compound bitwise xor)
|= (compound bitwise or)

# Microcontroller Fundamentals