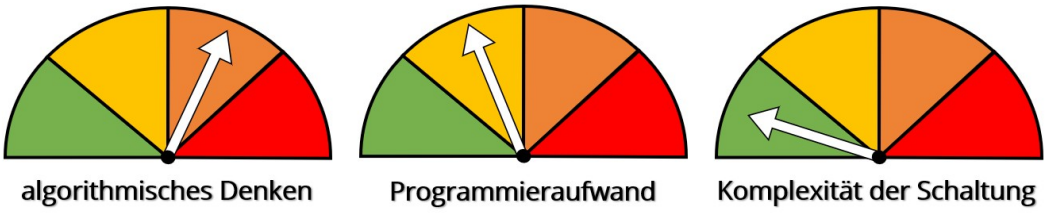
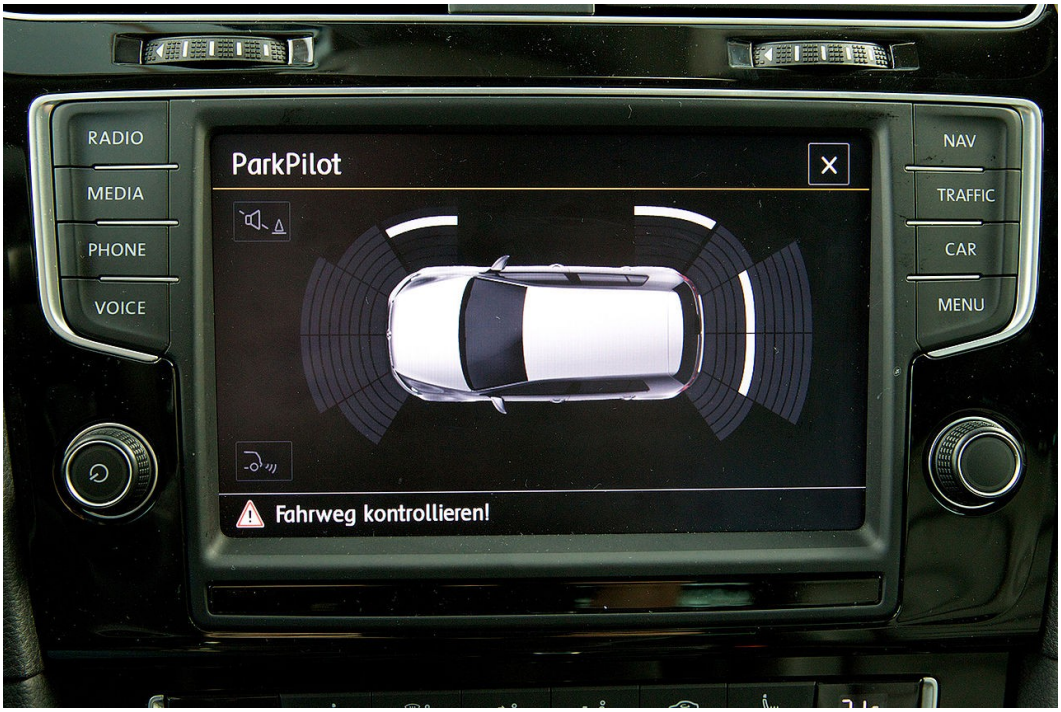
TECHNISCHE UNIVERSITÄT DRESDEN

MicroPython

Station 2 | Einparkhilfe





Nozilla, „Parking assist“, wikipedia.org , CC BY-SA 3.0

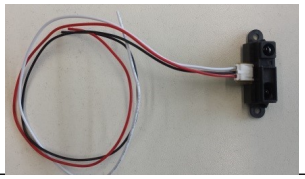
Professur für Didaktik der Informatik

UM WAS WIRD ES IN DIESER STATION GEHEN?

Jeder von euch, der demnächst in der Fahrschule seine Runden dreht, wird ein „Schreckensszenario“ kennenlernen: rückwärts einparken! Man fährt mehrmals vor und zurück, dreht ein und aus und am Ende steht man doch schief.
Was für ein Glück, dass in immer mehr Fahrzeugen Einparkhilfen eingebaut sind, die durch ihr drängendes und immer schneller werdendes Piepen vor dem Zusammenstoß warnen.
Aber wie funktioniert so eine Einparkhilfe? Wie misst das Auto den Abstand? Wie wird das Piepen erzeugt? Das wirst du in dieser Station lernen.

BENÖTIGTE BAUTEILE

Zusätzlich zum ESP32-Board und dem Steckbrett brauchst du folgende Bauteile:

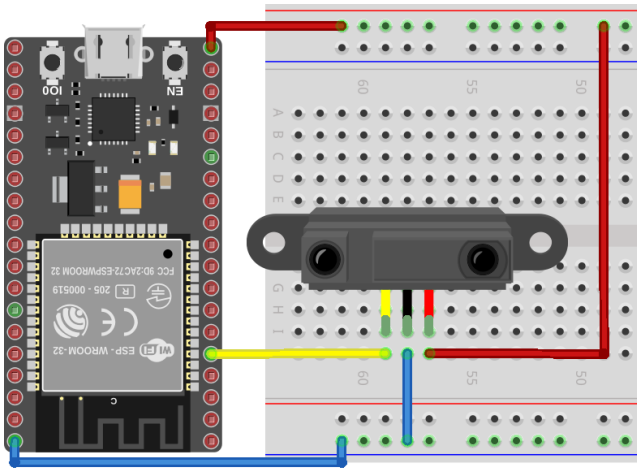
Piezo-lautsprecher	Diesen Lautsprecher brauchst du, um das Piepsignal zu erzeugen. Beachte, dass der Lautsprecher ein kurzes und ein langes Bein hat!	
Infrarot-Distanz-Mess-Sensor	Durch diesen Sensor kannst du Abstände zwischen 10 und 80cm messen.	

AUFGABE 1 – ABSTAND MESSEN

Als ersten Schritt sollst du den Abstand zwischen dem Infrarot-Sensor und einem Gegenstand messen. Befolge dabei die folgenden Punkte.

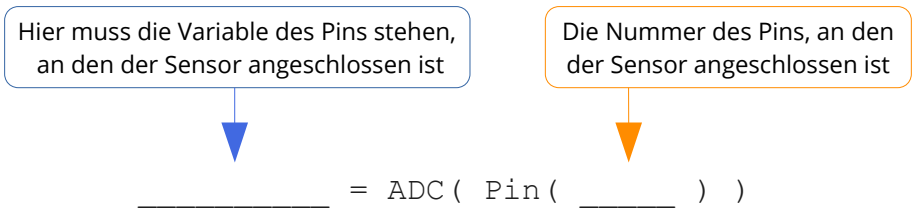
1. Bibliotheken importieren
   Wie im Einführungsbeispiel, benötigst du auch hier wieder die Bibliothek `machine`, um mit dem Board über MicroPython kommunizieren zu können. Weißt du noch, wie man Bibliotheken importiert? Falls nicht, schaue einfach noch einmal in der Einführungsstation nach oder frage einen Betreuer.

2. Sensor anschließen



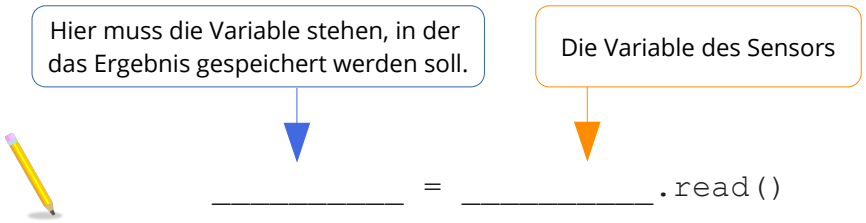
   Verbinde das rote Kabel des Infrarotsensors mit 3.3V und das schwarze Kabel mit GND (0V). Das weiße (im Bild gelbe) Kabel verbindest du mit einem Pin, der analoge Signale messen kann. Benutze am besten, wie im Bild gezeigt, **Pin 34**

3. Messvariable definieren
   Wie in der Einführungsstation, musst du das Board auch hier wissen lassen, an welchem Pin die Abstandsmessung stattfinden soll. Definiere dafür eine Variable. Das funktioniert für einen analogen Pin fast genauso, wie für einen digitalen. Setze einfach zusätzlich den Befehl `ADC()` um die Angabe des Pins:



4. Messung durchführen

   Nun ist der Sensor einsatzbereit! Definiere eine Variable, auf der der Messwert des Sensors gespeichert werden soll. Der Befehl zur Messung lautet `read()`:



   Füge den Befehl in eine `while True` – Schleife ein, damit er wiederholt wird, solange das Programm läuft. Falls du dir nicht sicher bist, wie man eine Schleife einfügt, schaue noch einmal in der Einführungsstation nach.

5. Messergebnis ausgeben lassen

   Natürlich sollen die gemessenen Werte in der Konsole unserer Programmierumgebung ausgegeben werden. Wie im Einführungsbeispiel verwenden wir dafür auch hier wieder den `print()` – Befehl.
   Was musst du in die Klammern des Befehls schreiben, damit die Messwerte des Sensors auf der Konsole ausgegeben werden?

6. Pausen zwischen den Messungen

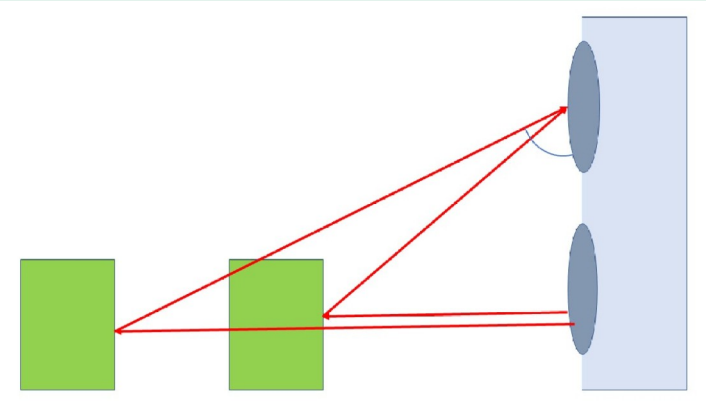
   Wie du vielleicht schon gemerkt hast, sieht man innerhalb kürzester Zeit sehr viele Messergebnisse. Aus diesem Grund ist es sinnvoll, kurze Pausen zwischen den Messungen einzufügen. Den Befehl dafür hast du schon öfter verwendet. Weißt du noch, wie er heißt?

   _____

   Achtung! Denke daran, den Befehl aus der Bibliothek `time` zu importieren, bevor du ihn benutzt. Falls du Hilfe brauchst, orientiere dich daran, wie du die Bibliothek `machine` importiert hast.

WIE FUNKTIONIERT EIN INFRAROT-DISTANZ-MESS-SENSOR?

Um den Abstand von einem Gegenstand messen zu können, hat der Sensor zwei „Augen“. Ein Auge sendet einen infraroten (für uns unsichtbaren) Lichtstrahl aus. Wenn das Licht auf einen Gegenstand trifft, wird es reflektiert. Das funktioniert ungefähr so, wie wenn du Sonnenlicht mit einem Spiegel reflektierst. Das reflektierte Licht kommt dann beim anderen Auge des Sensors an. Um den Abstand zu ermitteln, erkennt der Sensor, in welchem Winkel das reflektierte Licht ankommt. Je kleiner der Abstand, desto größer ist der Winkel. Diese Information wird über einen analogen Pin an den ESP32 weitergegeben. Und zwar, indem die Spannung verändert wird. Je größer die Spannung am Pin, desto weiter weg ist der Gegenstand vom Sensor.



AUFGABE 2 – MESSERGEBNIS UMRECHNEN

Beim Testen solltest du gemerkt haben, dass die Zahl, je weiter der Sensor vom Gegenstand entfernt ist, kleiner wird. Die Zahlen an sich sehen aber noch komisch aus. Der ESP32 misst den Abstand nicht in Zentimeter oder Meter. Er hat seine eigene Einheit. In dieser Aufgabe sollst du herausfinden, welches Messergebnis für welchen Abstand steht.

WIE MISST DER INFRAROTSENSOR?

Der Infrarotsensor ist mit dem Board über einen Schaltkreiskreis verbunden. Die Kabel sind die einzige Verbindung über die sie Informationen austauschen können. Der Sensor schickt das Ergebnis seiner Messung also als elektrisches Signal. Je näher ein Gegenstand ist, desto größer ist die Spannung, die der ESP32 am Ausgang des Sensors messen kann. Der Ausgang des Sensors nimmt dabei Werte zwischen 0V und 5V an.

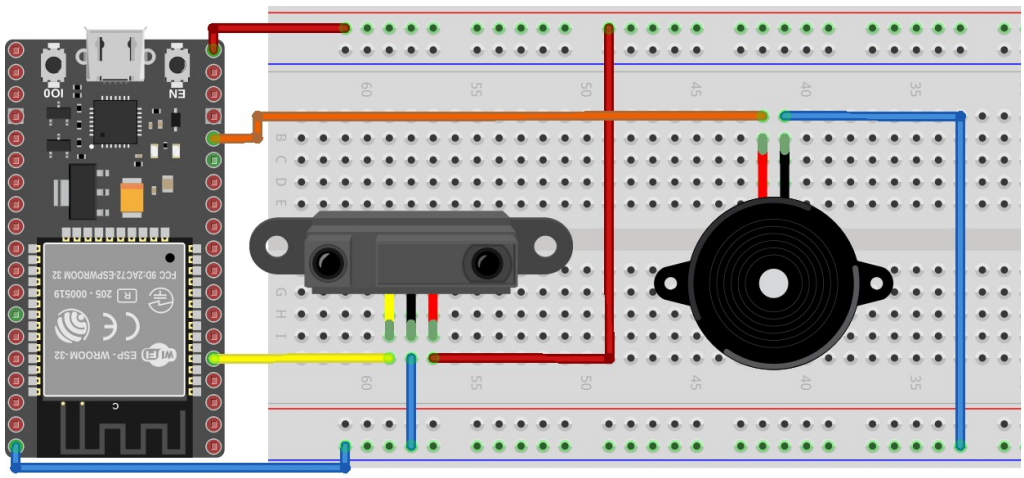
1. <u>Abstände messen</u>
   Um den Abstand in Zentimeter umzurechnen, musst du die Messergebnisse genau beobachten. Der Sensor kann Abstände zwischen 10 und 80cm messen. Stelle ein Stück Papier in den unterschiedlichen Abständen vom Sensor entfernt hin (10cm, 20cm, etc.). Beobachte für jeden Abstand, welche Messergebnisse in der Konsole angezeigt werden. Sie können sehr unterschiedlich sein. Schreibe das ungefähre Ergebnis in die Tabelle:

10cm	20cm	30cm	40cm	50cm	60cm	70cm	80cm

AUFGABE 3 – TON ERZEUGEN

Bisher hast du dem Sensor dazu gebracht den Abstand zu messen und du kannst die Ergebnisse auf dem seriellen Monitor einem ungefähren Abstand zuordnen. In dieser Aufgabe lässt du den Lautsprecher einen Ton ausgeben, wenn der Abstand zum Gegenstand zu klein wird.

1. Lautsprecher anschließen



Zum Anschließen des Lautsprechers brauchst du zwei besondere Kabel (siehe Bild). In der Abbildung ist das blaue Kabel an das kürzere Beinchen angeschlossen und wird mit GND verbunden. Das lange Beinchen wird mit einem Pin verbunden.

2. Pin-Variable definieren und Pin verbinden

Definiere eine Variable, die speichert, an welchem Pin der Lautsprecher angeschlossen ist. Da der Lautsprecher ein Signal *ausgeben* soll, müssen wir den Pinmodus anpassen:

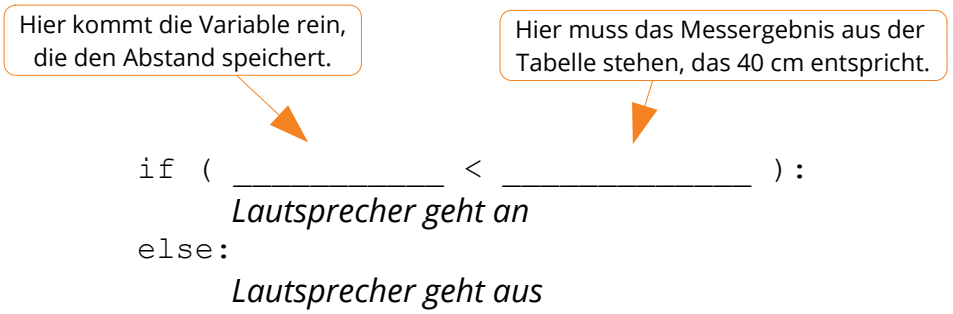
Die Variable des Lautsprechers

Der Pin, an den der Lautsprecher angeschlossen ist.

```
__________ = Pin( _____, Pin.OUT )
```

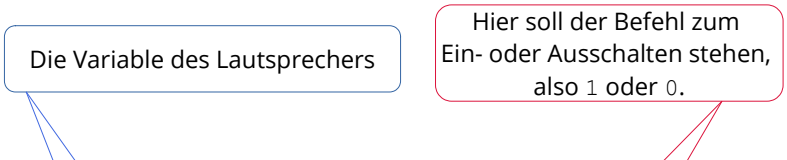
3. Wenn und sonst

Wenn der Abstand kleiner als 40 cm ist, soll der Lautsprecher angehen.
Sonst soll er nicht an sein.
Um das zu programmieren, musst du folgenden Ausdruck benutzen. Du hast ihn schon einmal benutzt. Füge ihn in die `while True` – Schleife ein.




```
if ( _____________ < _______________ ):
        Lautsprecher geht an
else:
        Lautsprecher geht aus
```

Jetzt fehlen noch die Befehle zum Angehen, bzw. Ausgehen. Dazu muss einfach der Lautsprecher ein-, bzw. ausgeschaltet werden. Das funktioniert ähnlich, wie das Ein- und Ausschalten einer LED.



zum Einschalten: `_____________.value( ______ )`

zum Ausschalten: `_____________.value( ______ )`

4. Installieren und testen

Installiere das Programm auf dem ESP32 und teste es mit einem Gegenstand. Der Lautsprecher sollte angehen, wenn der Gegenstand weniger als 40cm entfernt ist und ausgehen, wenn der Gegenstand wieder weiter weg ist.

AUFGABE 4 – PIEPEN

Super! Durch deine Einparkhilfe weißt du, ob ein Gegenstand näher ist, als 40cm. Als Autofahrer ist es aber wichtig genauere Informationen zu bekommen. Vor allem auf engen Parkplätzen muss man anderen Gegenständen oft nahe kommen. Je näher man einem Gegenstand kommt, desto schneller sollte dann das Piepen werden. Das wirst du in dieser Aufgabe programmieren.

1. Piepen programmieren

   Bisher hast du mit einem durchgehenden Ton gearbeitet. Ein Piepen entsteht, wenn ein Ton immer angeschaltet und kurz danach wieder ausgeschaltet wird. Also wie beim Blinken einer LED. Bringe die folgenden Befehle in die richtige Reihenfolge, um ein Piepen zu programmieren.

   _____
   _____
   _____
   _____

   A `sleep(0.3)`

   B `sleep(0.3)`

   C `_____.value( _____ )`

   D `_____.value( _____ )`

2. Abstände festlegen

   Um verschiedene Geschwindigkeiten für unterschiedliche Abstände zu programmieren, brauchst du wieder **Sonst wenn** – Ausdrücke. Füge die Ausdrücke zwischen den bereits geschriebenen `if` - und `else` – Teilen ein.

```
if ( _____ < _____ ):
    Lautsprecher geht an
elif ( _____ < _____ ):
    mittleres Piepen
elif ( _____ < _____ ):
    langsames Piepen
else:
    Lautsprecher geht aus
```

An diese Stellen muss die Variable stehen, die den Abstand speichert

Setze hier die unterschiedlichen Messergebnisse aus der Tabelle ein. Bspw. Die, die 30, 20cm entsprechen.

3. Schnelligkeit des Piepens

   Je näher der Gegenstand kommt, desto schneller soll das Piepen werden. Um die Geschwindigkeit des Piepens zu verändern musst du nur eine Kleinigkeit im Code ändern. Welchen Befehl musst du anpassen?

   _____

   Kopiere die Piep-Befehle in die neuen Sonst wenn – Ausdrücke. Verändere dann die Geschwindigkeit des Piepens so, dass du ein mittleres und ein langsames Piepen erhältst.

4. Installieren und testen

   Installiere das Programm auf dem ESP32. Teste die Piepgeschwindigkeiten, indem du einen Gegenstand unterschiedlich weit entfernt hältst.

Du hast jetzt eine Einparkhilfe programmiert, wie es sie auch in einem Auto gibt.