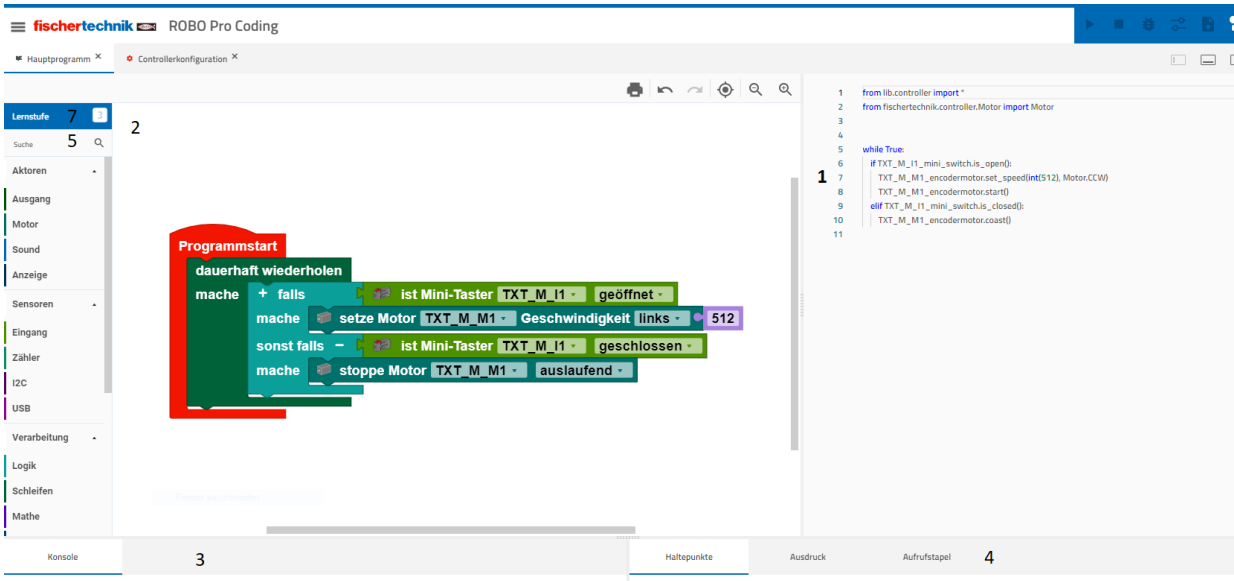
ROBO pro coding

fischertechnik

https://docs.fischertechnik-cloud.com/books/robo-pro-coding-de/

Allgemeine Benutzung

ROBO Pro Coding ist eine Entwicklungsumgebung zur Programmierung des neuen TXT 4.0 Controllers und damit zur Programmierung von fischertechnik-Modellen. ROBO Pro Coding ist verfügbar für Windows, macOS, Linux, iOS und Android und kann aus diesen Quellen bezogen werden.



Der abgebildete Code könnte für dieses Modell (links) genutzt werden.Die Weboberfläche von ROBO Pro Coding hat den folgenden Aufbau (die Ziffern verweisen auf die gekennzeichneten Bereiche in der Abbildung):

1. Hier steht das, was in Blöcken programmiert wurde, in Python
2. Das ist das Feld in das die Blöcke hineingezogen werden um zu programmieren
3. In der Konsole wird das Ablaufen des Programms dokumentiert
4.
    - Haltepunkte werden für das Debuggen von Programmen genutzt. Sie definieren bestimmte Stellen im Code, bei dehnen das Programm angehalten wird. Diese werden über den Editor gesetzt und bestehen aus dem Dateinamen + Nummer der Zeile.
    - Unter Ausdruck kann man sich Variableninhalte anschauen.
    - Der Aufrufstapel zeigt die Verschachtelung der Aufrufe. Insbesondere bei Einsatz von Funktionen ist das hilfreich.
5. Hier findet man alle Blöcke zum Programmieren sortiert in die Kategorien Aktoren, Sensoren und Verarbeitung
6. Diese Zeile wird separat unter Kopfzeile erläutert
7. Hier kann man das Lernniveau an den Nutzer anpassen. Je nach Niveau werden z.B. nur die allerwichtigsten Blöcke angezeigt.

Kopfzeile

In der Kopfzeile befinden sich (v.l.n.r.) die Bedienungselemente für das Starten des Programms, das Stoppen des Programms, das Starten des Debugger, den Aufruf des Schnittstellentests, das Hochladen des aktuellen Programms und das Verbinden mit dem Controller.

Starten von Programmen

Möchte man sein Programm von ROBO Pro Coding aus starten klickt man auf dieses Symbol.

Stoppen von Programmen

Möchte man sein Programm stoppen, bevor es von selbst endet tippt man auf dieses Symbol.

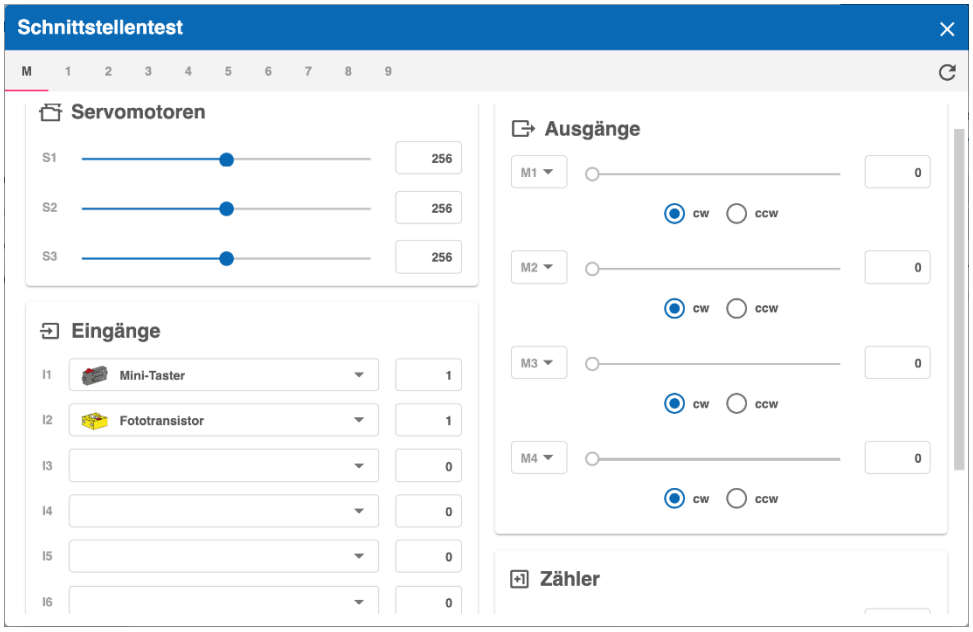
Der Debugger

Der Debugger soll das Finden von Fehlern im Programm erleichtern. Über das Menü in der oberen rechten Ecke kann man den Code Schritt für Schritt

durchgehen und so einen Programmdurchlauf simulieren. Das Programm reagiert dabei trotzdem auf Interaktion mit Sensoren, wie z.B. das drücken eines Tasters. Während man das Programm durchläuft, kann man links im großen Feld sehen in welchem Block man sich befinden und rechts an welcher Stelle im Python-Code.

Der Schnittstellentest

Klickt man auf das Symbol, mit den drei Reglern, öffnet sich dieses Fenster:



Hier kann man alle angeschlossenen Geräte sehen. Bei Aktoren kann man kann man über die Schieberegler überprüfen ob sie funktionieren. Bei den Sensoren wird der gemessene Wert angegeben (bei dem Taster steht z.B. eine 1, weil er gedrückt ist). Über das Dropdown-Menü (kleines Dreieck) kann angegeben werden, was angeschlossen ist.

Hochladen von Programmen

Hat man ein Programm geschrieben und möchte, dass es autark auf dem Controller läuft, kann man es über dieses Symbol auf den Controller laden. Es taucht dann unter den Dateien auf dem Controller auf. Tippt man dort auf den Programmnamen, landet man in den Unterorder des Programms. Hier findet sich eine Datei, die "main.py" heißt. Klicke auf diese und der "open"-Knopf unten rechts sollte grün aufleuchten. Drücken den "open"-Knopf und es öffnet sich ein weiteres Menü mit vier Optionen:

1. Laden: Lade das Programm dann kannst du es vom Homebildschirm aus starten
2. Auto load: Das Programm wird beim nächsten Starten des Controller automatisch wieder geladen

3. Auto start: Das Programm wird beim nächsten Starten des Controller automatisch wieder geladen und direkt gestartet
4. Delete project: Das Programm wird vom Controller gelöscht

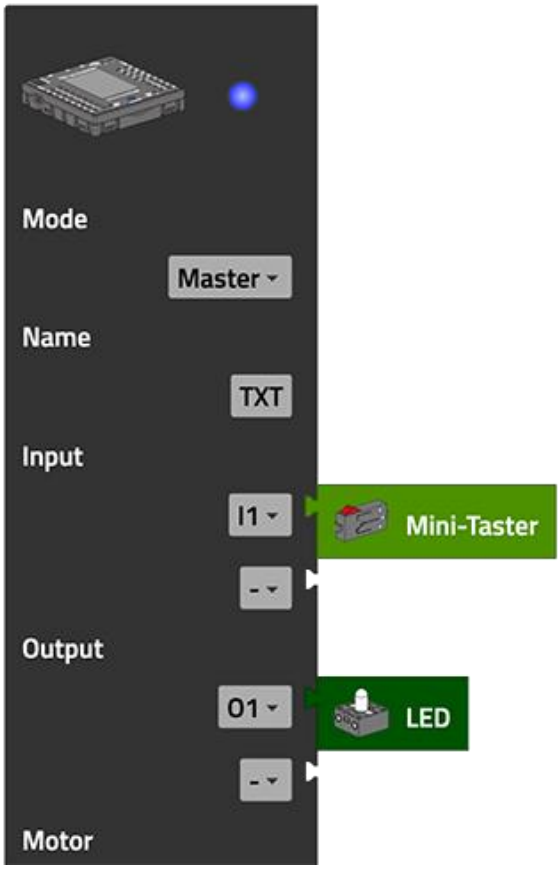
Verbinden mit dem Controller

Wie man sich mit dem Controller verbindet wurde hier erklärt.

Controllerkonfiguration

Um Bauteile, wie Sensoren und Aktoren, im Programm nutzen zu können, muss man sie an den Controller anschließen und das nicht nur physisch, sondern auch in der Software. Die Controllerkonfiguration ist immer automatisch geöffnet und kann oben direkt neben dem Hauptprogramm erreicht werden.

Im linken Bereich werden jetzt der Controller und sämtliche mögliche Bauteile, die sich an den Controller anschließen lassen, angezeigt. Den Controller zieht man auf die Programmierfläche. Anschließend kann man die gewünschten Bauteile per Drag and Drop an den Controller anschließen.



Speichern von Programmen

Wenn du ein Programm speichern möchtest klicke auf die drei Striche oben links auf der ROBO Pro Coding Website. Tippe hier auf die Option "Exportieren".

Jetzt kannst du auswählen ob du dein Programm lokal auf deinem Gerät oder bei GitLab speichern möchtest.

Lokal: Nachdem du den Namen deines Programms eingegeben hast klicke einfach auf "Exportieren" und die Programmdatei ist auf deinem Gerät.

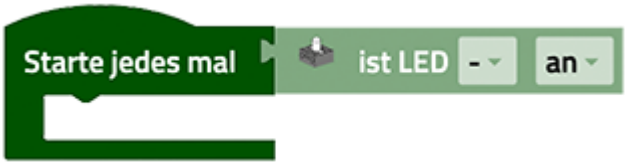
GitLab: Nachdem du den Namen deines Programm eingegeben hast und den angegebenen Schritten gefolgt bist, um den persönlichen Zugriffstoken einzugeben, klicke einfach auf "Exportieren" und die Programmdatei ist in deinem GitLab-Account gespeichert.

Aktoren

Ausgang

Der Starte jedes mal-Block

Der **Starte jedes mal-Block** bietet die Möglichkeit ein Programm ablaufen zulassen, wenn eine Bedingung erfüllt ist. Er funktioniert also ähnlich wie eine Fallunterscheidung wird aber nicht nur einmal durchlaufen, sondern jedes mal, wenn die Bedingung erfüllt ist, während des gesamten Ablauf des Programms. Der **Starte jedes mal-Block**:



Ist eine Abkürzung für folgendes Konstrukt:

Man kann in den **Starte jedes mal-Block** der Kategorie Ausgänge alle Bedingungen aus eben dieser Kategorie einsetzten.

Hinweis: Der Programmabschnitt innerhalb des Starte jedes mal-Block sollte kurz gehalten werden und keine blockierenden Aufrufe oder Endlosschleifen enthalten, so dass dieser Teil des Programms schnell abgearbeitet werden kann.

LEDs

Setzen

Mit den Blöcken **setze LED ...** und **setze LED Helligkeit ...** kann man die LED an- und ausstellen beziehungsweise ihre Helligkeit auf einen bestimmten Wert (von 0 bis 512) setzen.

Abrufen

Mit dem Block **hole LED Helligkeit** lässt sich die Helligkeit einer LED abrufen und als Wert weiterverarbeiten.

Abfragen

Mit den Blöcken **ist LED ...** und **ist LED Helligkeit ...** kann man die Aktivität beziehungsweise die Helligkeit einer LED als Bedingung nutzen. Im Beispiel wir die Helligkeit der LED auf 512 gesetzt, sofern sie nicht schon diese Helligkeit hat.

Motoren

Das Symbol auf den Motorblöcken steht stellvertretend für alle Motoren, die nicht Encoder- oder Servomotoren sind.

Setzen

Mit den Block **setze Motorgeschwindigkeit auf ()** ... kann man die Geschwindigkeit eines Motors auf einen bestimmten Wert (von 0 bis 512) setzen.

Abrufen

Mit dem Block **hole Motorgeschwindigkeit** lässt sich die Geschwindigkeit eines Motors abrufen und als Wert weiterverarbeiten.

Abfragen

Mit den Blöcken **läuft Motor** und **ist Motorgeschwindigkeit ...** kann man die Aktivität beziehungsweise die Geschwindigkeit eines Motor als Bedingung nutzen.

Stoppen

Mit dem Block **stoppe Motor ...** ist es möglich einen Motor zu stoppen.

Kompressor

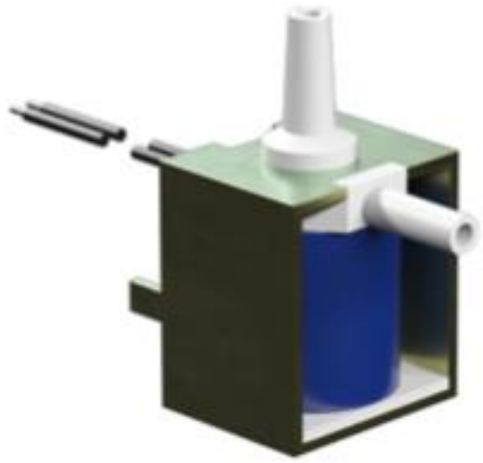
Setzen

Mit den Block **setze Kompressor ()** kann man den Kompressor ein- oder ausschalten.

Abfragen

Mit den Block **ist Kompressor ()** kann man die Aktivität eines Kompressors als Bedingung nutzen.

Magnetventil



Setzen

Mit den Block **setze Magnetventil ()** kann man das Magnetventil ein- oder ausschalten. Hier bedeutet "ein", dass das Ventil offen ist und "aus", dass das Ventil geschlossen ist.

Abfragen

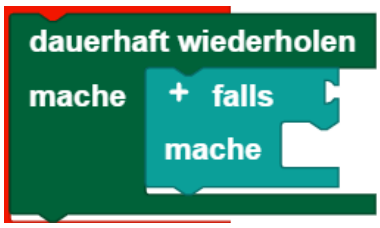
Mit den Block **ist Magnetventil ()** kann man die Aktivität eines Magnetventils als Bedingung nutzen.

Motor

Der Starte jedes mal-Block

Der **Starte jedes mal-Block** bietet die Möglichkeit ein Programm ablaufen zulassen, wenn eine Bedingung erfüllt ist. Er funktioniert also ähnlich wie eine Fallunterscheidung wird aber nicht nur einmal durchlaufen, sondern jedes mal, wenn die Bedingung erfüllt ist, während des gesamten Ablauf des Programms. Der **Starte jedes mal-Block**:

Ist eine Abkürzung für folgendes Konstrukt:



Man kann in den **Starte jedes mal-Block** der Kategorie Motor alle Bedingungen aus eben dieser Kategorie einsetzten.

Hinweis: Der Programmabschnitt innerhalb des Starte jedes mal-Block sollte kurz gehalten werden und keine blockierenden Aufrufe oder Endlosschleifen enthalten, so dass dieser Teil des Programms schnell abgearbeitet werden kann.

Motor

Das Symbol auf den Motorblöcken steht stellvertretend für alle Motoren, die nicht Encoder- oder Servomotoren sind.

Setzen

Mit den Block **setze Motorgeschwindigkeit auf ()** ... kann man die Geschwindigkeit eines Motors auf einen bestimmten Wert (von 0 bis 512) setzen. Über das Dropdown-Menü (kleines Dreieck) kann die Drehrichtung gewählt werden.

Abrufen

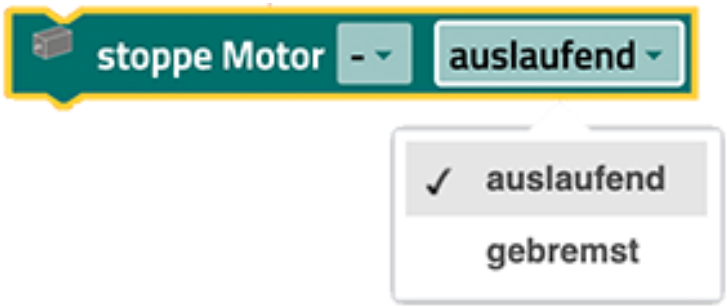
Mit dem Block **hole Motorgeschwindigkeit** lässt sich die Geschwindigkeit eines Motors abrufen und als Wert weiterverarbeiten.

Abfragen

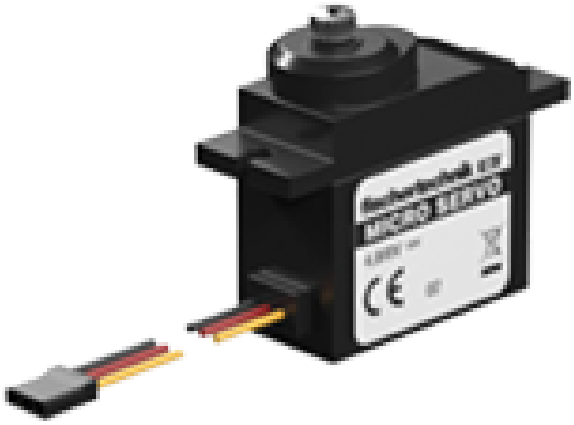
Mit den Blöcken **läuft Motor** und **ist Motorgeschwindigkeit ...** kann man die Aktivität beziehungsweise die Geschwindigkeit eines Motor als Bedingung nutzen.

Stoppen

Mit dem Block **stoppe Motor ()** ist es möglich einen Motor zu stoppen. Dabei bietet der Block **stoppe Motor ()** die Optionen, einen Motor direkt oder auslaufend zu stoppen. Die gewünschte Option kann über das Dropdown-Menü (kleines Dreieck) ausgewählt werden:



Servomotor



Setzen

Mit den Block **setze Position auf ...** kann man die Position eines Servomotors auf einen bestimmten Wert (von 0-512) setzen. 0 und 512 sind die Werte für die maximale Auslenkung rechts und links. Bei dem Wert 256 steht der Servomotor dementsprechend in der Mitte.

Abrufen

Mit dem Block **rufe Position ab** lässt sich die Position eines Servomotors abrufen und als Wert weiterverarbeiten.

Encodermotor

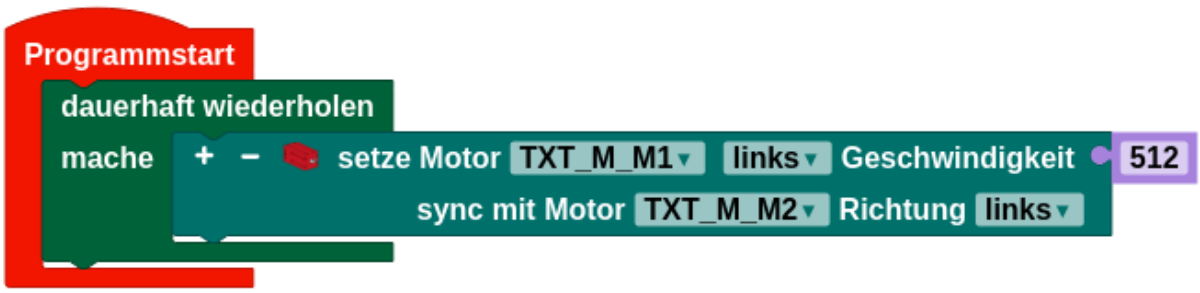
Der Encodermotor hat die gleichen Funktionen wie ein normaler Motor, bietet aber zusätzlich die Möglichkeit, die Umdrehungen zu zählen und mehrere Motoren synchron anzusteuern. Eine Umdrehung wird dabei in ~64 Schritte unterteilt.

Setzen

Mit dem Block

kann man die Geschwindigkeit eines Motors auf einen bestimmten Wert (von 0-512) setzen. Über das Dropdown-Menü (kleines Dreieck) kann die Drehrichtung gewählt werden. Zusätzlich kann man die Anzahl an Schritten eingeben, die der Motor zurücklegen soll. In diesem Beispiel dreht sich der Motor 100 Schritte, also eine und eine drittel Umdrehungen. Wie am Bespiel zu sehen hat dieser Block ein Pluszeichen, mit Hilfe dessen sich mehrere Motoren synchron ansteuern lassen. Es ist möglich Motoren am Master oder an einer Extension zu synchronisieren, eine übergreifenden Synchronisierung bspw. zwischen Motoren des Master und einer Extension ist nicht möglich.

Hinweis: Schnell aufeinanderfolgende Synchronisierungsaufrufe, wie sie z.B. durch eine Schleife möglich sind (siehe Beispiel), können die Synchronität beeinträchtigen oder sogar komplett verhindern.



Stoppen

Mit dem Block **stoppe Motor ...** stoppt man einen Motor. Möchte man mehrere Motoren gleichzeitig stoppen, kann man über das Plus links am Block bis zu drei weitere Motoren hinzufügen.

Abfragen

Der Block **hat Position erreicht** wird genutzt, um das Erreichen der Position als Bedingung zu nutzen. Mit Position ist hier die Endposition eines Encodermotors nach vollendeter Schrittweite gemeint.

Sound

Der TXT 4.0 Controller hat einen eingebauten Lautsprecher und bietet somit die Möglichkeit Sounds abzuspielen.

Der Starte jedes mal-Block

Der **Starte jedes mal-Block** bietet die Möglichkeit ein Programm ablaufen zulassen, wenn eine Bedingung erfüllt ist. Er funktioniert also ähnlich wie eine Fallunterscheidung wird aber nicht nur einmal durchlaufen, sondern jedes mal, wenn die Bedingung erfüllt ist, während des gesamten Ablauf des Programms. Der **Starte jedes mal-Block**:

Ist eine Abkürzung für folgendes Konstrukt:

Man kann in den **Starte jedes mal-Block** der Kategorie Sound alle Bedingungen aus eben dieser Kategorie einsetzten.

Hinweis: Der Programmabschnitt innerhalb des Starte jedes mal-Block sollte kurz gehalten werden und keine blockierenden Aufrufe oder Endlosschleifen enthalten, so dass dieser Teil des Programms schnell abgearbeitet werden kann.

Abspielen

Vorinstallierte Audiodateien

Mit dem folgenden Block kann man einen von 29 vorinstallierten Sounds abspielen. Die gewünschte Audiodatei kann über das Dropdown Menü (kleines Dreieck) ausgewählt werden. Außerdem ist es möglich, den Ton in Dauerschleife abzuspielen. Dafür muss man das Kästchen hinter dem Dauerschleife-Symbol ankreuzen.

Eigene Audiodateien

Möchte man einen eigenen Sound abspielen, kann man den Block

nutzen. Um seinen eigenen Sound in den Block einzubetten muss man:

1. Mit dem Controller Verbunden sein
2. Die IP-Adresse des Controllers in den Browser eingeben (hierbei muss die IP gewählt werden, die auch zum Verbinden mit dem Controller genutzt wurde)
3. Auf der aufgerufenen Seite USER: ft, PASSWORD: fischertechnik eingeben
4. Ordner sounds öffnen und dort über das Plus die gewünschte Audiodatei auf den Controller laden (wichtig: die Audiodatei muss im wav-Format vorliegen)
5. Im ROBO Pro Coding-Block unter Pfad "./dateiname.wav" angeben

Auch hier gibt es die Option, den Sound in Dauerschleife abzuspielen.

Abfragen

Um abzufragen, ob eine Audiodatei abgespielt wird, nutzt man den Block **gibt Ton wieder**. Dieser kann als Bedingung im Programm genutzt werden.

Stoppen

Um einen Ton zu stoppen, wird einfach der Block **stoppe Tonwiedergabe** im Programm verwendet.

Anzeige

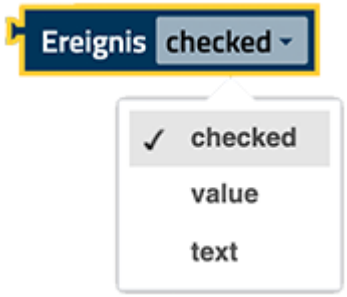
Mit den Blöcken der Kategorie Display lässt sich der Bildschirm des TXT 4.0 Controllers gestalten und nutzbar machen. Dies geschieht in zwei Schritten:

1. Konfigurieren, das heißt
    - Eine neue Datei der Kategorie Display öffnen, über das Seiten Symbol mit dem Plus oben links
    - die gewünschten Elemente auf den gerasterten Bereich ziehen (er stellt den konfigurierbaren Teil des Displays dar)
    - bei Bedarf Spezifikationen anpassen.
2. Programmieren, das heißt
    - Im Hauptprogramm mit den Blöcken der Kategorie Display die Wirkung von Interaktion mit dem Display programmieren.

Blöcke

Ereignisabfrage

Der Block **Ereignis ()** ruft den Rückgabewert eines Elements ab. Dieser Block kann nur in den Ereignisprogrammen genutzt werden. In diesen Ereignisprogrammen bezieht sich der Block automatisch auf das Ereignis in dessen Programm er verwendet wird. Der geeignete Typ für den Rückgabewert, kann über das Dropdown-Menü (kleines Dreieck) gewählt werden:



Beschriftungsfeld

Mit dem Element Beschriftungsfeld kann man einen Text auf dem Bildschirm platzieren. Das Symbol im Anzeigekonfigurator ist das Etikett. Zieht man dieses Symbol in den gerasterten Bereich, öffnet sich rechts ein Fenster. Hier kann unter Inspektor

- die Größe des Beschriftungsfelds in Pixeln,
- die Position des Beschriftungsfelds in Pixeln (auf dem angegebenen Punkt liegt die obere linke Ecke des Textfeldes),
- der Name des Beschriftungsfelds und
- der Inhalt des Beschriftungsfelds (dieser Text wird bei Start des Displays abgebildet)

festgelegt werden.

Mit dem Block **setze Beschriftungsfeld Text ...** lässt sich der abgebildete Text im Laufe des Programms ändern.

Eingaben

Das Element **Eingabe** erlaubt es, dass Nutzer*innen über den Controller Text eingeben. Das zugehörige Symbol im Anzeigekonfigurator ist das "T" Zeichen. Zieht man dieses Symbol in den gerasterten Bereich, öffnet sich rechts ein Fenster. Hier kann unter Inspektor

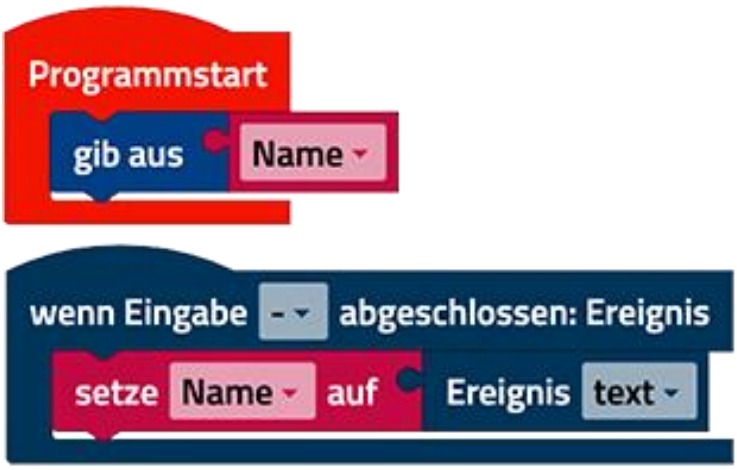
- die Größe des Eingabefeldes in Pixeln,
- die Position des Eingabefeldes in Pixeln (auf dem angegebenen Punkt liegt die obere linke Ecke des Eingabefeldes),
- der Name des Eingabefeldes und
- der Inhalt des Eingabefeldes (dieser Text wird bei Start des Displays abgebildet)

festgelegt werden.

Mit dem Block **setze Eingabefeld Text ...** lässt sich der abgebildete Text im Laufe des Programms ändern.

Eingabe-Programm

Das Eingabe-Programm läuft ab, wenn eine Eingabe abgeschlossen wurde. Es wird separat vom Hauptprogramm geschrieben. Variablen funktionieren global über beide Programme hinweg. Das Eingabe-Programm läuft im Block **wenn Eingabe abgeschlossen** ab. Der **Ereignis ()**-Block wird im Eingabe-Programm auf "text" gesetzt. In diesem Beispiel wird die Variable **Name** auf den eingegebenen Text gesetzt, sie wird dann im Hauptprogramm genutzt, um den eingegebenen Text auszugeben:



Messinstrument

Die Messinstrument-Funktion kann Werte (keine Werte kleiner 1) darstellen. Das zugehörige Symbol im Displaykonfigurator ist die Skalierung. Zieht man dieses Symbol in den gerasterten Bereich, öffnet sich rechts ein Fenster. Hier kann unter Inspektor

- die Größe des Messinstruments in Pixeln,
- die Position des Messinstruments in Pixeln (auf dem angegebenen Punkt lieg die obere linke Ecke der Messlehre),
- der Name des Messinstruments,
- die Ausrichtung des Messinstruments
- der Wertebereich, den das Messinstrument darstellt, und
- der Wert des Messinstruments, der bei Start des Displays gezeigt wird

festgelegt werden.

Mit dem Block **setze Messinstrument auf Wert …** lässt sich das Messinstrument auf den eingegebenen Wert setzen. Dieser Wert sollte im vorher definierten Wertebereich liegen. Liegt der Wert außerhalb des Wertebereich, wird, je nachdem ob der Wert zu groß oder zu klein ist, eine der Grenzen des Wertebereichs dargestellt.

Statusanzeige

Der Statusindikator zeigt die Aktivität von etwas an. Je nach Status leuchtet er ("aktiv") oder tut dies nicht ("inaktiv"). Das Symbol im Displaykonfigurator ist eine leuchtende Diode. Zieht man dieses Symbol in den gerasterten Bereich, öffnet sich rechts ein Fenster. Hier kann unter Inspektor

- die Größe der Statusanzeige in Pixeln,
- die Position der Statusanzeige in Pixeln (auf dem angegebenen Punkt liegt die obere linke Ecke des Statusindikators),
- der Name der Statusanzeige,
- die Farbe der Statusanzeige und
- ob die Statusanzeige zu Beginn aktiv oder inaktiv sein soll,

festgelegt werden.

Mit dem Block **setze Statusanzeige aktiv ()** lässt sich die Statusanzeige aktivieren bzw. deaktivieren. Im Dropdown-Menü (kleines Dreieck) lässt sich wählen, ob die Statusanzeige auf aktiv oder inaktiv gesetzt werden soll.

Schieberegler

Der Schieberegler gibt Werte abhängig von seiner Position zurück. Die Position kann dabei vom Nutzer*innen über den Touchscreen verändert werden. Der Wert kann über den **Ereignis ()**-Block abgerufen werden, sobald der Schieberegler ruht. Der abgerufene Wert ist eine Dezimalzahl. Will man den Wert des Schiebereglers ganzzahlig haben, muss man den **runde**-Block

einsetzen. Das zugehörige Symbol für den Schieberegler ist der Strich mit dem Kreis. Zieht man dieses Symbol in den gerasterten Bereich, öffnet sich rechts ein Fenster. Hier kann unter Inspektor

- die Größe des Schiebereglers in Pixeln,
- die Position des Schiebereglers in Pixeln (auf dem angegebenen Punkt liegt dann die obere linke Ecke des Schiebereglers)
- der Name des Schiebereglers,
- die Aktivität des Schiebereglers,
- die Ausrichtung des Schiebereglers,
- den Wertebereich der über den Schieberegler abgedeckt wird und
- der Wert, auf dem der Regler bei Start des Displays steht

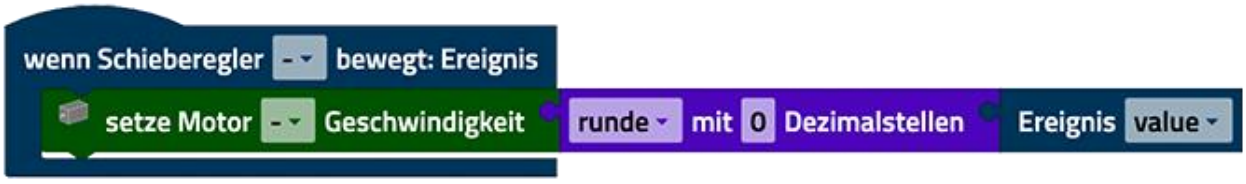
festgelegt werden.

Mit dem Block **Setze Schieberegler Wert ...** kann man den Schieberegler auf einen anderen Wert verschieben.

Mit **setze Schieberegler aktiviert ()** kann man die Aktivität über das Dropdown-Menü (kleines Dreieck) wechseln.

Schieberegler-Programm

Das Schieberegler-Programm läuft ab, nachdem der Schieberegler verschoben wurde. Es wird separat vom Hauptprogramm geschrieben. Variablen funktionieren global über beide Programme hinweg. Das Schieberegler-Programm läuft im Block **wenn Schieberegler bewegt** ab. Der **Ereignis ()**-Block wird im Schieberegler-Programm auf "value" gesetzt. In diesem Beispiel wird die Geschwindigkeit des Motors über den Schieberegler gesteuert. Der Wert des Schiebereglers muss gerundet werden, da der Motor nur ganze Zahlen als Drehzahl akzeptiert:



Schaltfläche

Die Schaltfläche ist ein beschriftetes Feld, das gedrückt werden kann. Drückt man die Schaltfläche, läuft das Schaltflächen-Programm ab, sobald sie wieder losgelassen wird. Das zugehörige Symbol für die Schaltfläche ist das Quadrat

mit der "OK" Beschriftung. Zieht man dieses Symbol in den gerasterten Bereich, öffnet sich rechts ein Fenster. Hier kann unter Inspektor

- die Größe der Schaltfläche in Pixeln,
- die Position der Schaltfläche in Pixeln (auf dem angegebenen Punkt liegt die obere linke Ecke des Knopfes),
- der Name der Schaltfläche,
- den Text, der auf der Schaltfläche steht und
- die Aktivität der Schaltfläche

festgelegt werden.

Mit dem Block **setze Schaltfläche aktiviert ()** kann man die Aktivität über das Dropdown-Menü (kleines Dreieck) wechseln.

Schaltflächen-Programm

Das Schaltflächen-Programm läuft ab, sobald die Schaltfläche nicht mehr gedrückt ist. Es wird separat vom Hauptprogramm geschrieben. Variablen funktionieren global über beide Programme hinweg. Das Schaltflächen-Programm läuft im Block **wenn Schaltfläche angeklickt** ab. Der **Ereignis ()**-Block kann im Schaltflächen-Programm nicht verwendet werden, da die Schaltfläche keinen Rückgabewert hat. In diesem Beispiel wird die LED aktiviert, wenn die Schaltfläche gedrückt wurde.

Schalter

Der Schalter kann zwei Positionen einnehmen und befindet sich immer in genau einer dieser beiden Positionen. Je nach Position gibt er **wahr** oder **falsch** zurück. Das zugehörige Symbol für den Schalter ist das Oval mit dem Punkt. Zieht man dieses Symbol in den gerasterten Bereich, öffnet sich rechts ein Fenster. Hier kann unter Inspektor

- die Größe des Schalters in Pixeln,
- die Position des Schalters in Pixeln (auf dem angegebenen Punkt liegt die obere linke Ecke des Schalters),
- der Name des Schalters,
- den Text, der neben dem Schalter steht,
- die Aktivität des Schalters und
- den Zustand in dem sich der Schalter bei Start des Programms befinden soll

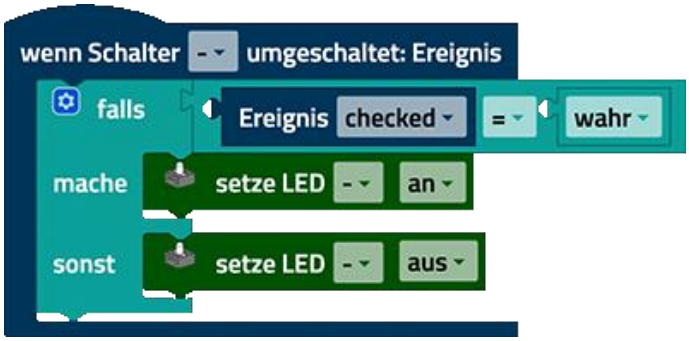
angepasst werden.

Der Block

übernimmt zwei Funktionen. Man kann entweder die Aktivität (enabled im Dropdown-Menü wählen) oder den Zustand (checked im Dropdown-Menü wählen) auf **wahr** oder **falsch** setzen.

Schalter-Programm

Das Schalter-Programm läuft jedes mal ab, wenn der Schalter umgelegt wird. Es wird separat vom Hauptprogramm geschrieben. Variablen funktionieren global über beide Programme hinweg. Das Schalter-Programm läuft im Block **wenn Schalter umgeschaltet** ab. Der **Ereignis ()**-Block wird im Schalter-Programm auf "checked" gesetzt, er gibt **wahr** oder **falsch** zurück. Dieses Beispielprogramm schaltet die LED ein, wenn der Schalter umgelegt ist, andernfalls wird die LED ausgeschaltet:



Kontrollkästchen

Das Kontrollkästchen kann zwei Zustände annehmen und befindet sich immer in genau einem dieser beiden. Je nach Zustand gibt es **wahr** oder **falsch** zurück. Das Symbol für das Kontrollkästchen ist das Quadrat mit dem Haken. Zieht man dieses Symbol in den gerasterten Bereich, öffnet sich rechts ein Fenster. Hier kann unter Inspektor

- die Größe des Kontrollkästchens in Pixeln,
- die Position des Kontrollkästchens in Pixeln (auf dem angegebenen Punkt liegt die obere linke Ecke des Kontrollkästchens),
- der Name des Kontrollkästchens,
- den Text, der neben dem Kontrollkästchen steht,

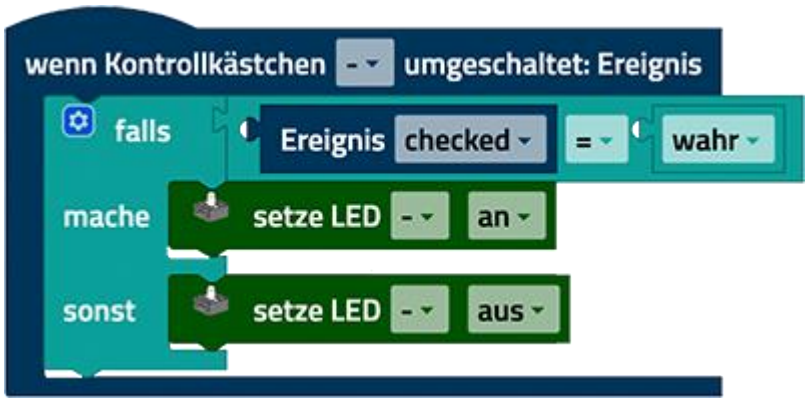
- die Aktivität des Kontrollkästchens und
- den Zustand in dem sich das Kontrollkästchen bei Start des Programms befinden soll

festgelegt werden.

Der folgende Block übernimmt zwei Funktionen. Über das Dropdown-Menü (kleines Dreieck) kann gewählt werden, welche man nutzt. Man kann entweder die Aktivität (enabled im Dropdown-Menü wählen) oder den Zustand (checked im Dropdown-Menü wählen) auf **wahr** oder **falsch** setzen.

Kontrollkästchen-Programm

Das Kontrollkästchen-Programm läuft jedes mal ab, wenn das Kontrollkästchen gedrückt wird. Es wird separat vom Hauptprogramm geschrieben. Variablen funktionieren über beide Programme hinweg. Das Kontrollkästchen-Programm läuft im Block **wenn Kontrollkästchen umgeschaltet** ab. Der **Ereignis** ()-Block wird im Schalter-Programm auf "checked" gesetzt, er gibt **wahr** oder **falsch** zurück. Dieses Beispielprogramm schaltet die LED ein, wenn das Kontrollkästchen angehakt ist, andernfalls wird die LED ausgeschaltet.

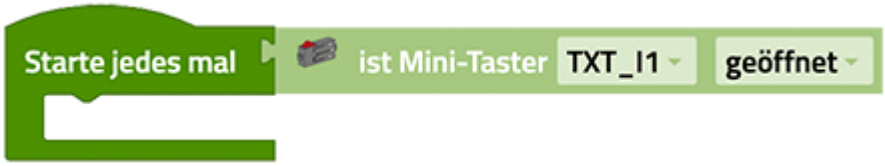


Sensoren

Eingang

Der Starte jedes mal-Block

Der **Starte jedes mal-Block** bietet die Möglichkeit ein Programm ablaufen zulassen, wenn eine Bedingung erfüllt ist. Er funktioniert also ähnlich wie eine Fallunterscheidung wird aber nicht nur einmal durchlaufen, sondern jedes mal, wenn die Bedingung erfüllt ist, während des gesamten Ablauf des Programms. Der **Starte jedes mal-Block**:



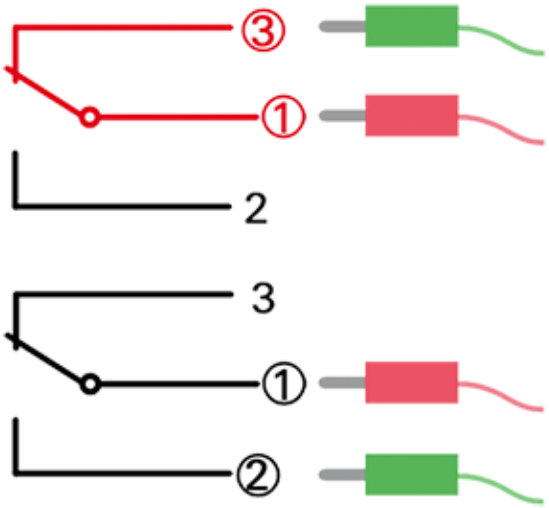
Ist eine Abkürzung für folgendes Konstrukt:

Man kann in den **Starte jedes mal-Block** der Kategorie Eingaben alle Bedingungen aus eben dieser Kategorie einsetzten.

Hinweis: Der Programmabschnitt innerhalb des Starte jedes mal-Block sollte kurz gehalten werden und keine blockierenden Aufrufe oder Endlosschleifen enthalten, so dass dieser Teil des Programms schnell abgearbeitet werden kann.

Taster

Der Taster ist ein digitaler Sensor, der "Es fließt Strom" von "Es fließt kein Strom" unterscheidet. Ob Strom fließt hängt dabei sowohl von der Verkabelung, als auch davon ab, ob der Taster gedrückt ist. Man kann den Taster also auf zwei verschieden Arte verwenden:



Als „Schließer“:
Kontakte 1 und 3 werden angeschlossen.

Taster gedrückt – Es fließt Strom
Taster nicht gedrückt – Es fließt kein Strom

Als „Öffner“:

Kontakte 1 und 2 werden angeschlossen.

Taster gedrückt – Es fließt kein Strom
Taster nicht gedrückt – Es fließt Strom

Abrufen

Mit **hole Mini-Taster Status** erhält man Information darüber, ob durch den Taster Strom fließt oder nicht. Fließt Strom, wird **1** zurückgegeben fließt kein Strom, **0**.

Abfragen

Um abzufragen, ob der Taster in einem bestimmten Zustand ist, wird der Block **ist Taster...** genutzt. Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt werden nach welchem Zustand gefragt wird. Dieser Block kann als Bedingung genutzt werden.

Ultraschallsensor

Der Ultraschallsensor wird genutzt, um Entfernungen zu messen.

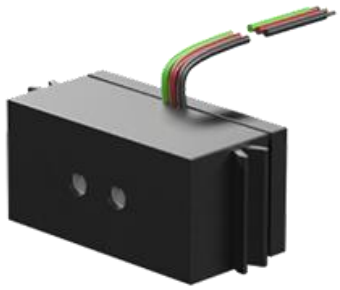
Abrufen

Mit **hole Ultraschallsensor Abstand** erhält man die Information, wie weit der Sensor vom nächsten Gegenstand entfernt ist. Der Abstand wird in cm zurückgegeben.

Abfragen

Um abzufragen, ob der Sensor einen bestimmten Abstand zum nächsten Gegenstand hat , wird der Block **ist Ultraschallsensor Abstand...** genutzt. Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt werden, wie der gemessene Abstand mit einem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >) . Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn der gemessen Abstand kleiner als 2 cm ist.

Farbsensor



Der Farbsensor sendet rotes Licht aus und misst, wie viel davon zurückreflektiert wird. Je nachdem, wie stark die Reflexion ist, gibt der Farbsensor Werte von 0 bis 2000 zurück. Er eignet sich gut, um vorher kalibrierte Farben zu erkennen.

Abrufen

Mit **hole Farbsensor Wert** erhält man die Information, wie stark eine Oberfläche das Licht reflektiert.

Abfragen

Um abzufragen, ob der Sensor eine bestimmte Farbe vor sich hat, wird der Block **ist Farbsensor Wert ...** genutzt. Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt werden, wie der gemessene Farbwert mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >) . Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn der gemessen Farbwert kleiner als 1000 ist.

IR-Spursensor

Der Infrarot-Spursensor ist ein digitaler Sensor zur Erkennung einer schwarzen Spur auf weißem Untergrund, der bei einem Abstand von 5-30 mm von Sensor zu Untergrund arbeitet.

Abrufen

Mit **hole IR-Spusensor Status** erhält man 0, wenn der Sensor keine Spur erkennt. Erkennt der Sensor eine Spur, wird 1 zurückgegeben

Abfragen

Um abzufragen, ob der IR-Spurensensor eine Spur erkennt, vergleicht man den aktuellen Spurstatus mit 0 oder 1. Hierzu eignet sich der Block **ist IR-Spusensor Status () ...** . Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt erden, wie der Spurstatus mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >). Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn der Spurstatus 0 ist.

Fototransistor

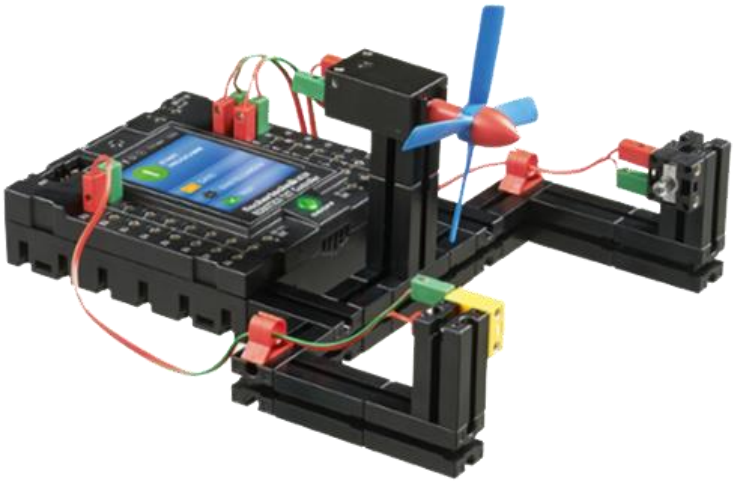
Der Fototransistor ist ein digitaler Sensor, der hell von dunkel unterscheidet.

Abrufen

Mit **hole Fototransistor Status** erhält man 0, wenn der Sensor kein Licht erkennt. Erkennt der Sensor ausreichend Licht, wird 1 zurückgegeben.

Abfragen

Um abzufragen, ob der Fototransistor hell oder dunkel erkennt, vergleicht man den Helligkeitsstatus mit 0 oder 1. Hierzu eignet sich der Block **ist Fototransistor Status ()**. Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt werden, ob hell oder dunkel abgefragt werden soll.



Eine mögliche Verwendung für einen Fototransistor ist in einer Lichtschranke, wie diesem Modell.

Fotowiderstand

Der Widerstand des Fotowiderstands sinkt, wenn er mehr Helligkeit ausgesetzt ist. Der ausgegebene Wert des Fotowiderstands ist also ein Maß für Helligkeit.

Abrufen

Mit **hole Fotowiderstand Wert** erhält man die Information, wie hell es ist. Je kleiner der ausgegebene Wert, desto heller ist es.

Abfragen

Um abzufragen, ob der Fotoresistor einen bestimmten Helligkeitswert misst, wird der Block **ist Fotowiderstand Wert () ...** genutzt Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt wie der Helligkeitswert mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >). Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn der Helligkeitswert kleiner als 3000 ist.

NTC-Widerstand

Der NTC-Widerstand ist ein nichtbinärer Temperatursensor. Sein elektrischer Widerstand sinkt, wenn die Temperatur steigt, und ist damit ein Maß für die Temperatur.

Abrufen

Mit **hole NTC-Widerstand ()** erhält man entweder einen Widerstandswert oder die daraus errechnete Temperatur. Was zurückgegeben werden soll kann über das Dropdown-Menü (kleines Dreieck) gewählt werden.

Abfragen

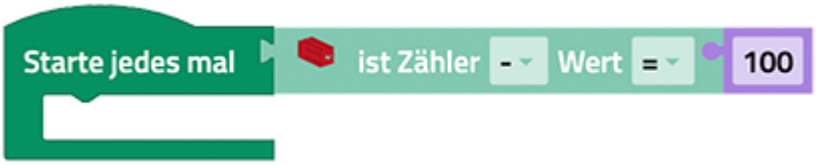
Um abzufragen, ob der NTC-Widerstand einen bestimmten Wert misst, wird der Block **ist NTC-Widerstand () ()** ... Über die Dropdown-Menüs (kleines Dreieck) kann gewählt werden, was und mit welchem Vergleichsoperator verglichen werden soll. Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn die Temperatur kleiner als 20 ist.

Zähler

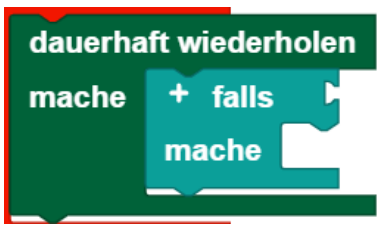
Der Encodermotor kann als Zähler eingesetzt werden. Gezählt wird die Anzahl seiner Umdrehungen, und zwar nicht nur, wenn er sich selber als Motor dreht, sondern auch, wenn er von außen mechanisch angetrieben wird.

Der Starte jedes mal-Block

Der **Starte jedes mal-Block** bietet die Möglichkeit ein Programm ablaufen zulassen, wenn eine Bedingung erfüllt ist. Er funktioniert also ähnlich wie eine Fallunterscheidung wird aber nicht nur einmal durchlaufen, sondern jedes mal, wenn die Bedingung erfüllt ist, während des gesamten Ablauf des Programms. Der **Starte jedes mal-Block**:



Ist eine Abkürzung für folgendes Konstrukt:



Man kann in den **Starte jedes mal-Block** der Kategorie Zähler alle Bedingungen aus eben dieser Kategorie einsetzten.

Hinweis: Der Programmabschnitt innerhalb des Starte jedes mal-Block sollte kurz gehalten werden und keine blockierenden Aufrufe oder Endlosschleifen enthalten, so dass dieser Teil des Programms schnell abgearbeitet werden kann.

Abrufen

Mit **hole Zähler Wert** erhält man den vom Zähler gezählten Wert.

Abfragen

Um abzufragen, ob der Zähler einen bestimmten Wert gezählt hat, wird dieser Block

genutzt. Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt wie der gezählte Wert mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >).

Zurücksetzen

Der Zähler beginnt wieder bei 0, wenn der Block **setze Zähler zurück** ausgeführt wird.

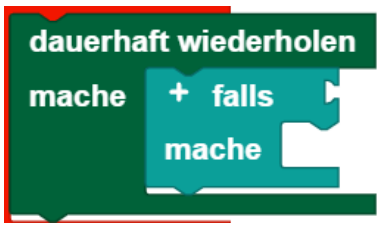
I2C

Die in diesem Kapitel beschriebenen I2C-Sensoren werden über ein geeignetes Flachbandkabel mit dem TXT 4.0 Controller verbunden.

Der Starte jedes mal-Block

Der **Starte jedes mal-Block** bietet die Möglichkeit ein Programm ablaufen zulassen, wenn eine Bedingung erfüllt ist. Er funktioniert also ähnlich wie eine Fallunterscheidung wird aber nicht nur einmal durchlaufen, sondern jedes mal, wenn die Bedingung erfüllt ist, während des gesamten Ablauf des Programms. Der **Starte jedes mal-Block**:

Ist eine Abkürzung für folgendes Konstrukt:



Man kann in den **Starte jedes mal-Block** der Kategorie I2C alle Bedingungen aus eben dieser Kategorie einsetzten.

Hinweis: Der Programmabschnitt innerhalb des Starte jedes mal-Block sollte kurz gehalten werden und keine blockierenden Aufrufe oder Endlosschleifen enthalten, so dass dieser Teil des Programms schnell abgearbeitet werden kann.

Kombisensor

Der Kombisensor vereint die drei Funktion Beschleunigungssensor, Gyroskop und Kompasssensor in einem Bauteil.

Beschleunigungssensor

Abrufen

Mit **hole Kombisensor Beschleunigung in ()** erhält man die Beschleunigung in einer Raumrichtungen. Die gewünschte Raumrichtung kann über das Dropdown-Menü (kleines Dreieck) gewählt werden. Die Beschleunigung wird in g angegeben.

Abfragen

Um abzufragen, ob man eine bestimmte Beschleunigung misst, wird der Block **ist Kombisensor Beschleunigung in () () …** genutzt. Über die Dropdown-Menüs (kleines Dreieck) kann ausgewählt wie die Beschleunigung mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >) und welche Raumrichtung abgefragt werden soll. Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn die Beschleunigung in x-Richtung größer als 10 ist.

Gyroskop

Abrufen

Mit **hole Kombisensor Rotation in ()** erhält man die Rotation in einer Raumrichtungen. Die gewünschte Raumrichtung kann über das Dropdown-Menü (kleines Dreieck) gewählt werden. Die Rotation wird in °/s angegeben.

Abfragen

Um abzufragen, ob man eine bestimmte Winkelgeschwindigkeit misst, wird der Block **ist Kombisensor Rotation in () () ...** genutzt. Über die Dropdown-Menüs (kleines Dreieck) kann ausgewählt wie die Rotation mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >) und welche Raumrichtung abgefragt werden soll. Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn die Rotation in x-Richtung größer als 10 ist.

Kompasssensor

Abrufen

Mit **hole Kombisensor Magnetfluss in ()** erhält man den magnetischen Fluss in einer Raumrichtungen. Die gewünschte Raumrichtung kann über das Dropdown-Menü (kleines Dreieck) gewählt werden. Der magnetische Fluss wird in µT angegeben.

Abfragen

Um abzufragen, ob man einen bestimmten magnetischen Fluss misst, wird der Block **ist Kombisensor Magnetfluss in () () ...** genutzt. Über die Dropdown-Menüs (kleines Dreieck) kann ausgewählt wie der magnetische Fluss mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >) und welche Raumrichtung abgefragt werden soll. Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung

genutzt werden. Im Beispiel wird der Motor gestoppt, wenn der magnetische Fluss in x-Richtung größer als 10 ist.

Umweltsensor

Der Umweltsensor vereint die vier Funktionen Luftqualitätssensor, Luftfeuchtigkeitssensor, Barometer und Thermometer in einem Bauteil.

Luftqualitätssensor

Abrufen

Mit dem Block **hole Umweltsensor Luftqualität als ()** kann man die Luftqualität messen. Über das Dropdown-Menü (kleines Dreieck) kann gewählt werden, ob die Luftqualität als Zahlenwert (von 0 bis 500) oder als Text zurückgegeben werden soll.

Abfragen

Um abzufragen, ob man eine bestimmte Luftqualität misst, wird der Block **ist Umweltsensor Luftqualität () ...** genutzt. Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt wie die Luftqualität mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >). Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn die Luftqualität größer als 10 ist.

Barometer

Abrufen

Mit dem Block **hole Umweltsensor Luftdruck** kann man den Luftdruck messen.

Abfragen

Um abzufragen, ob man einen bestimmten Luftdruck misst, wird der Block **ist Umweltsensor Luftdruck () ...** genutzt. Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt wie der Luftdruck mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >). Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn der Luftdruck größer als 10 ist.

Thermometer

Abrufen

Mit dem Block **hole Umweltsensor Temperatur** kann man die Temperatur messen.

Abfragen

Um abzufragen, ob man eine bestimmte Temperatur misst, wird der Block **ist Umweltsensor Temperatur () ...** genutzt. Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt wie die Temperatur mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >). Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn die Temperatur größer als 10 ist.

Luftfeuchtigkeitssensor

Abrufen

Mit dem Block **hole Umweltsensor Luftfeuchtigkeit** kann man die Luftfeuchtigkeit messen.

Abfragen

Um abzufragen, ob man eine bestimmte Luftfeuchtigkeit misst, wird der Block **ist Umweltsensor Luftfeuchtigkeit () ...** genutzt. Über das Dropdown-Menü (kleines Dreieck) kann ausgewählt wie die Luftfeuchtigkeit mit dem eingegebenen Wert verglichen werden soll (<, ≤, =, ≠, ≥, >). Der Vergleichswert wird in das Zahlenfeld am Ende des Blocks eingegeben. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn die Luftfeuchtigkeit größer als 10 ist.

USB

USB

Über den USB-Anschluss kann die Kamera mit integriertem Mikrophon angeschlossen werden. Kamera und Mikrophon werden hier getrennt betrachtet. Um die Funktionen der Kamera zu nutzen, muss man sie erst im Kamera Konfigurator konfigurieren. Wie man zum Kamerakonfigurator gelangt, wird hier erklärt.

Kamera

Die Kamera kann insbesondere als Bewegungsdetektor, als Farbdetektor, als Balldetektor und als Liniendetektor genutzt werden.

Bewegungsdetektor

Um die Kamera als Bewegungsdetektor zu nutzen, muss man im Kamerakonfigurator das Männchen-Symbol in die gerasterte Fläche ziehen, dann öffnet sich rechts ein Fenster in dem man unter Inspektor

- den Pixelbereich, in dem auf Bewegung überprüft wird,
- die Position dieses Bereichs (auf dem angegebenen Punkt liegt die obere linke Ecke des Bereichs),
- den Namen des Bewegungsdetektors und
- die Toleranz

festlegen kann.

Bewegungsdetektor Programm

Das Bewegungsdetektor-Programm läuft ab, wenn eine Bewegung erkannt wurde. Es wird separat vom Hauptprogramm geschrieben. Variablen funktionieren global über beide Programme hinweg. Das Eingabe-Programm läuft im Block **wenn Bewegung erkannt** ab.

Farbdetektor

Um die Kamera als Farbdetektor zu nutzen, muss man im Kamerakonfigurator das Pipetten-Symbol in die gerasterte Fläche ziehen, dann öffnet sich rechts ein Fenster in dem man unter Inspektor

- den Pixelbereich, in dem auf Farbe überprüft wird,
- die Position dieses Bereichs (auf dem angegebenen Punkt liegt die obere linke Ecke des Bereichs),
- den Namen des Farbdetektors und
- den Kontrast

festlegen kann.

Abrufen

Mit **hole Farbe als ()** erhält man die erkannte Farbe im Hexadezimal oder im RGB Format. Das Format kann über das Dropdown-Menü (kleines Dreieck) eingestellt werden.

Abfragen

Um abzufragen, ob der Detektor eine Farbe wahrgenommen hat, wird der Block **ist Farbe detektiert** genutzt. Dieser Block kann als Bedingung genutzt werden.

Um abzufragen, ob der Detektor eine bestimmte Farbe wahrnimmt wird dieser Block

genutzt. Mit dem Block kann man die aufgenommene Farbe mit einer eingegeben vergleichen. Über das Dropdown-Menü (kleines Dreieck) kann gewählt werden ob die eingestellte Farbe gleich oder ungleich der gefilmten Farbe seien soll. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn die aufgenommene Farbe rot ist.

Farbdetektor Programm

Das Farbdetektor-Programm läuft ab, wenn eine Farbe erkannt wurde. Es wird separat vom Hauptprogramm geschrieben. Variablen funktionieren global über beide Programme hinweg. Das Eingabe-Programm läuft im Block **wenn Farbe erkannt** ab.

Balldetektor

Um die Kamera als Balldetektor zu nutzen, muss man im Kamerakonfigurator das Bälle-Symbol in die gerasterte Fläche ziehen, dann öffnet sich rechts ein Fenster in dem man unter Inspektor

- den Pixelbereich, in dem auf Bälle überprüft wird,
- die Position dieses Bereichs (auf dem angegebenen Punkt liegt die obere linke Ecke des Bereichs),
- den Namen des Balldetektors,
- den Bereich, in dem der Durchmesser des Balle liegt,
- den Bereich der x-Achse,
- die Farbe des Balles und
- die Farbtoleranz

festlegen kann.

Abrufen

Mit **hole () des Balls** erhält man x-Position, y-Position, Radius oder Durchmesser, des Balles.

Abfragen

Um abzufragen, ob der Detektor einen Ball wahrgenommen hat, wird der Block **ist Ball detektiert** genutzt. Dieser Block kann als Bedingung genutzt werden.

Um abzufragen, ob der Detektor einen Ball mit einer bestimmten x-Position, y-Position, Radius oder Durchmesser wahrnimmt wird dieser Block

genutzt. Mit dem Block kann man Spezifikationen, des aufgenommenen Balles, mit einem eingegeben Wert vergleichen. Über die Dropdown-Menüs (kleines Dreieck) kann gewählt werden, was und mit welchem Vergleichsoperator verglichen werden soll. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn der Durchmesser, des aufgenommenen Balles, 5 ist.

Balldetektor Programm

Das Balldetektor-Programm läuft ab, wenn eine Farbe erkannt wurde. Es wird separat vom Hauptprogramm geschrieben. Variablen funktionieren global über beide Programme hinweg. Das Eingabe-Programm läuft im Block **wenn Ball erkannt** ab.

Liniendetektor

Um die Kamera als Liniendetektor zu nutzen, muss man im Kamerakonfigurator das Symbol mit den Punkten auf einer Linie in die gerasterte Fläche ziehen, dann öffnet sich rechts ein Fenster in dem man unter Inspektor

- den Pixelbereich, in dem auf Linien überprüft wird,
- die Position dieses Bereichs (auf dem angegebenen Punkt liegt die obere linke Ecke des Bereichs),
- den Namen des Liniendetektors,
- die Anzahl an Linien, die erkannt werden sollen, und
- den Bereich, in dem die Breite der Linie(n) liegt

festlegen kann.

Abrufen

Mit **hole () der Linie ()** erhält man Position oder Breite einer der maximal fünf Linien ab.

Mit **hole Farbe der Linie () als ()** erhält die Farbe einer Linie im Hexadezimal oder im RGB Format ausgeben lassen. Das Format kann über das Dropdown-Menü (kleines Dreieck) eingestellt werden.

Abfragen

Um abzufragen, ob der Detektor eine Linie wahrgenommen hat, wird der Block **ist Linie detektiert** genutzt. Dieser Block kann als Bedingung genutzt werden.

Um abzufragen, ob der Detektor eine Linie mit einer bestimmten Position oder Breite wahrnimmt wird dieser Block

genutzt. Mit dem Block kann man Spezifikationen, der aufgenommenen Linie(n), mit einem eingegeben Wert vergleichen. Über die Dropdown-Menüs (kleines Dreieck) kann gewählt werden, was und mit welchem Vergleichsoperator verglichen werden soll. Dieser Block kann als Bedingung genutzt werden. Im Beispiel wird der Motor gestoppt, wenn die Breite, der aufgenommenen Linie, kleiner als 2 ist.

Um abzufragen, ob der Detektor eine Linie mit einer bestimmten Farbe wahrnimmt wird dieser Block

genutzt. Mit dem Block kann man die aufgenommene Linienfarbe mit einer eingegeben vergleichen. Über das Dropdown-Menü (kleines Dreieck) kann gewählt werden ob die eingestellte Farbe gleich oder ungleich der gefilmten Farbe seien soll. Dieser Block kann als Bedingung genutzt werden.

Liniendetektor Programm

Das Liniendetektor-Programm läuft ab, wenn eine oder mehrere Linien erkannt wurden. Es wird separat vom Hauptprogramm geschrieben. Variablen

funktionieren global über beide Programme hinweg. Das Eingabe-Programm läuft im Block **wenn Linien erkannt** ab.

Mikrofon

Das in der Kamera integrierte Mikrofon kann als Lautstärkedetektor genutzt werden.

Der Starte jedes mal-Block

Der **Starte jedes mal-Block** bietet die Möglichkeit ein Programm ablaufen zulassen, wenn eine Bedingung erfüllt ist. Er funktioniert also ähnlich wie eine Fallunterscheidung wird aber nicht nur einmal durchlaufen, sondern jedes mal, wenn die Bedingung erfüllt ist, während des gesamten Ablauf des Programms. Der **Starte jedes mal-Block**:

Ist eine Abkürzung für folgendes Konstrukt:

Man kann in den **Starte jedes mal-Block** der Kategorie Mikrofron alle Bedingungen aus eben dieser Kategorie einsetzten.

Hinweis: Der Programmabschnitt innerhalb des Starte jedes mal-Block sollte kurz gehalten werden und keine blockierenden Aufrufe oder Endlosschleifen enthalten, so dass dieser Teil des Programms schnell abgearbeitet werden kann.

Lautstärkedetektor

Abrufen

Mit **Mikrofon Lautstärke** erhält man die Lautstärke in Dezibel.

Abfragen

Um abzufragen, ob der Lautstärkedetektor eine bestimmte Lautstärke wahrnimmt, wird dieser Block

genutzt. Mit dem Block kann man die aufgenommene Lautstärke mit einer eingegebenen vergleichen. Über das Dropdown-Menü (kleines Dreieck) kann gewählt werden, mit welchem Vergleichsoperator verglichen werden soll. Dieser Block kann als Bedingung genutzt werden.

Verarbeitung

Bedingungen

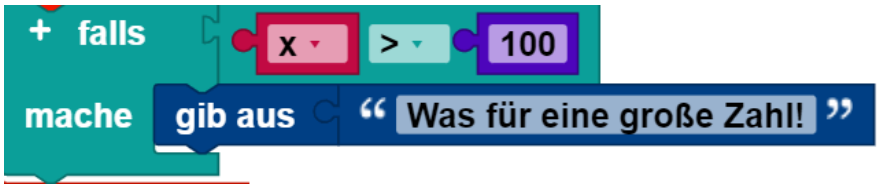
Bedingte Anweisungen sind zentral für die Programmierung. Sie machen es möglich, Fallunterscheidungen zu formulieren wie:

- Wenn es einen Weg nach links gibt, biege links ab.
- Wenn Punktzahl = 100, drucke "Gut gemacht!".

Blöcke

wenn-Blöcke

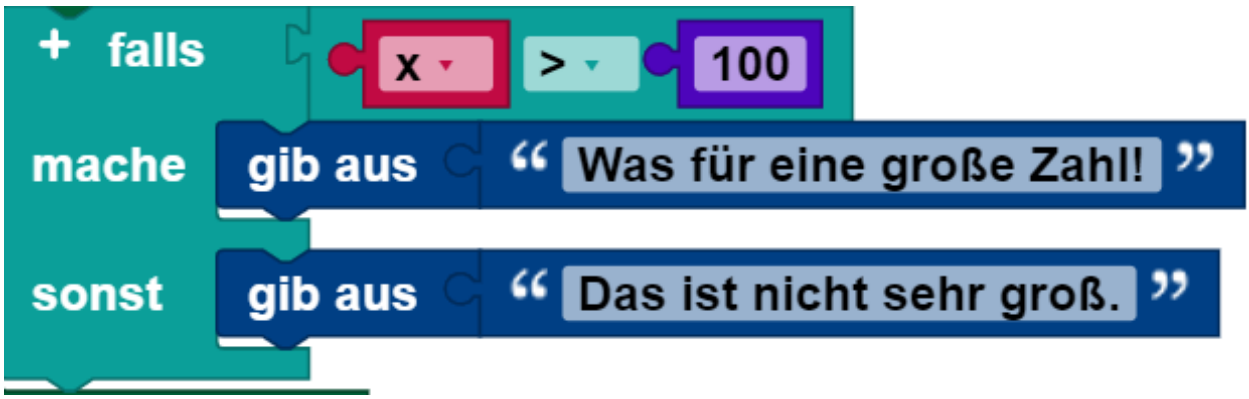
Die einfachste Bedingung ist ein **wenn**-Block:



Wenn dieser ausgeführt wird, wird der Wert der Variable **x** mit 100 verglichen. Wenn er größer ist, wird "Was für eine große Zahl!" ausgegeben. Andernfalls passiert nichts.

wenn-sonst-Blöcke

Es ist auch möglich, anzugeben, dass etwas passieren soll, wenn die Bedingung nicht wahr ist, wie in diesem Beispiel:



Wie beim vorherigen Block wird "Was für eine große Zahl!" ausgegeben, wenn **x** > 100 ist. Andernfalls wird "Das ist nicht sehr groß." angegeben.

Ein **wenn**-Block einen **sonst**-Abschnitt haben, aber nicht mehr als einen.

wenn-sonst-wenn-Blöcke

Es ist auch möglich, mehrere Bedingungen mit einem einzigen **wenn**-Block zu testen, indem **sonst-wenn**-Klauseln hinzugefügt werden:

Der Block prüft zuerst, ob **x** > 100 ist, und gibt "Was für eine große Zahl!" aus, wenn das der Fall ist. Ist dies nicht der Fall, prüft er weiter, ob **x** = 42 ist. Wenn ja, gibt er "Das ist meine Glückszahl!" aus. Andernfalls passiert nichts.

Ein **wenn**-Block kann eine beliebige Anzahl von **sonst-wenn**-Abschnitten haben. Die Bedingungen werden von oben nach unten ausgewertet, bis eine erfüllt ist oder bis keine Bedingung mehr übrig sind.

wenn-sonst-wenn-sonst-Blöcke

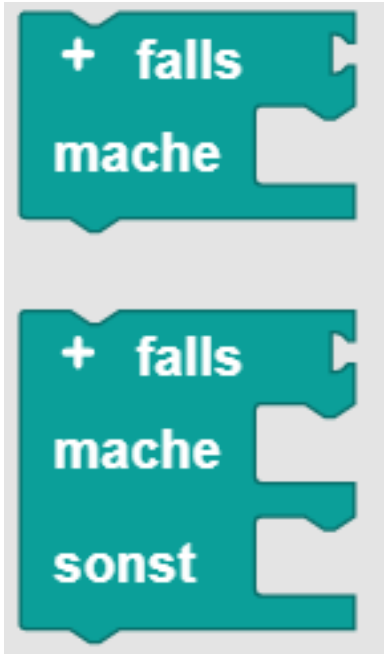
wenn-Blöcke können sowohl **sonst-wenn** als auch **sonst**-Abschnitte haben:

Der **sonst**-Abschnitt garantiert, dass eine Aktion ausgeführt wird, auch wenn keine der vorherigen Bedingungen wahr ist.

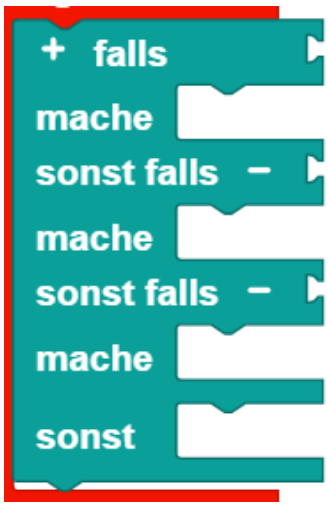
Ein **sonst**-Abschnitt kann nach einer beliebigen Anzahl von **sonst-wenn**-Abschnitten auftreten, einschließlich Null, dann erhält man einen ganz normalen **wenn-sonst**-Block.

Blockmodifikation

In der Werkzeugleiste erscheint nur der einfache **wenn**-Block und der **wenn-sonst**-Block:



Um **sonst-wenn** - und **sonst**-Klauseln hinzuzufügen, klickst du auf das (+) Symbol. Mit (-) Symbol lassen sich **sonst-wenn** -Klauseln wieder entfernen:



Beachte, dass die Formen der Blöcke das Hinzufügen einer beliebigen Anzahl von **sonst-wenn**-Unterblöcken erlauben, aber nur bis zu einen **wenn**-Block.

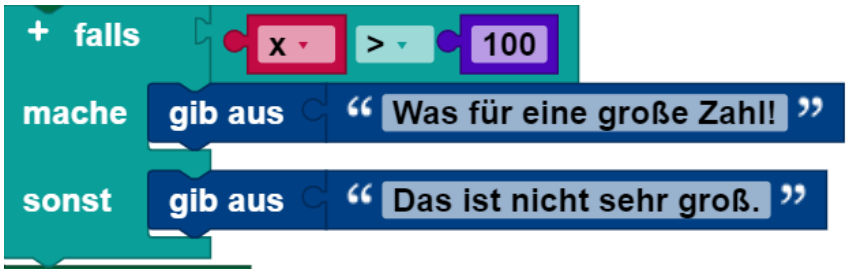
Logik

Boolesche Logik ist ein einfaches mathematisches System, das zwei Werte hat:

- **wahr**
- **falsch**

Logikblöcke in ROBO Pro Coding sind in der Regel dafür da, Bedingungen und Schleifen zu kontrollieren.

Hier ein Beispiel:



Wenn der Wert der Variable x größer als 100 ist, ist die Bedingung wahr und der Text "Was für eine große Zahl!" wird ausgegeben. Ist der Wert von x nicht größer als 100, ist die Bedingung falsch und "Das ist nicht sehr groß." wird ausgegeben. Boolesche Werte können auch in Variablen gespeichert werden und an Funktionen weitergegeben werden, genauso wie Zahlen, Text und Listenwerte.

Blöcke

Wenn ein Block einen Booleschen Wert als Eingabe erwartet, wird eine fehlende Eingabe als **falsch** interpretiert. Nicht-Boolesche Werte können nicht

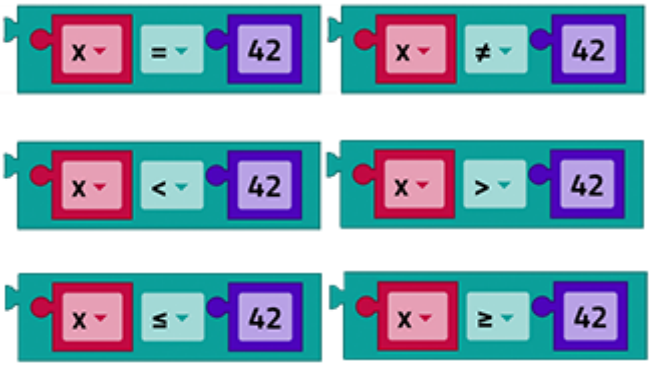
direkt dort eingefügt werden, wo Boolesche Werte erwartet werden, obwohl es möglich (aber nicht ratsam) ist, einen nicht-Booleschen Wert in einer Variablen zu speichern und diese dann in die Bedingungseingabe einzufügen. Diese Methode wird nicht empfohlen, und ihr Verhalten kann sich in zukünftigen Versionen von ROBO Pro Coding ändern.

Werte

Ein einzelner Block mit einer Dropdown-Liste, die entweder **wahr** oder **falsch** angibt, kann verwendet werden, um einen Booleschen Wert abzurufen:

Vergleichsoperatoren

Es gibt sechs Vergleichsoperatoren. Jedem werden zwei Eingaben (normalerweise zwei Zahlen) übergeben und der Vergleichsoperator gibt **wahr** oder **falsch** zurück, je nachdem, wie die Eingaben miteinander verglichen werden.



Die sechs Operatoren sind: gleich, nicht gleich, kleiner als, größer als, kleiner als oder gleich, größer als oder gleich.

Logische Operatoren

Der **und**-Block gibt dann und nur dann **wahr** zurück, wenn seine beiden Einganswerte wahr sind.

Der **oder**-Block gibt **wahr** zurück, wenn mindestens einer seiner beiden Einganswerte wahr ist.

nicht

Der **nicht**-Block wandelt eine boolesche Eingabe in ihr Gegenteil um. Zum Beispiel ist das Ergebnis von:

falsch.

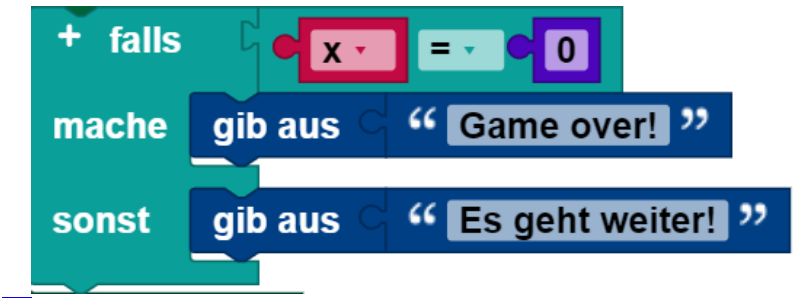
Wenn keine Eingabe erfolgt, wird der Wert **wahr** angenommen, so dass der folgende Block den Wert **falsch** erzeugt:

Es wird jedoch nicht empfohlen, eine Eingabe leer zu lassen.

dreier Operator

Der dreier Operator verhält sich wie ein Miniatur-**wenn-sonst**-Block. Er nimmt drei Eingangswerte entgegen der erste Eingangswert ist die zu testende boolesche Bedingung, der zweite Eingangswert ist der Wert, der zurückgegeben werden soll, wenn der Test **wahr** ergibt, der dritte Eingangswert ist der Wert, der zurückgegeben werden soll, wenn der Test falsch ergibt. Im folgenden Beispiel wird die Variable **Farbe** auf rot gesetzt, wenn die Variable **x** kleiner als 10 ist, andernfalls wird die Variable **Farbe** auf grün gesetzt.

Ein dreier Block kann immer durch einen **wenn-sonst**-Block ersetzt werden. Die folgenden zwei Beispiele sind genau gleich.



Schleifen

Der Bereich "Steuerung" enthält Blöcke, die steuern, ob andere Blöcke, die in ihrem Inneren platziert sind, ausgeführt werden. Es gibt zwei Arten von Steuerungsblöcken: wenn-sonst-Blöcke (die auf einer eigenen Seite beschrieben werden) und Blöcke, die steuern, wie oft ihr Inneres ausgeführt wird. Letztere werden Schleifen genannt, da ihr Inneres, auch als Schleifenkörper oder Körper bezeichnet,(möglicherweise) mehrfach wiederholt wird. Jeder Durchlauf einer Schleife wird als Iteration bezeichnet.

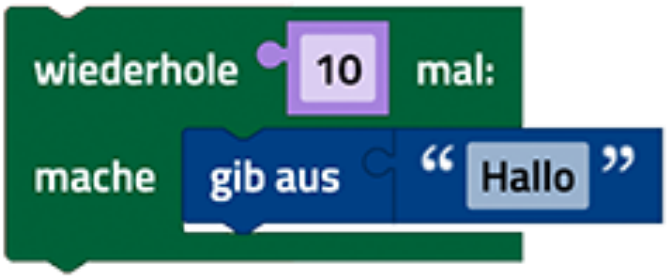
Blöcke zur Erstellung von Schleifen

dauerhaft wiederholen

Der **dauerhaft wiederholen**-Block führt den Code in seinem Körper solange aus, bis das Programm endet.

wiederhole

Der **wiederhole**-Block führt den Code in seinem Körper, so häufig wie angegeben aus. Der folgende Block gibt zum Beispiel zehnmal "Hallo!" aus:



wiederhole-solange

Stelle dir ein Spiel vor, bei dem ein Spieler einen Würfel wirft und alle geworfenen Werte addiert, solange die Summe kleiner als 30 ist. Die folgenden Blöcke implementieren dieses Spiel:

1. Eine Variable namens **insgesamt** erhält einen Anfangswert von 0.

2. Die Schleife beginnt mit einer Überprüfung, ob **insgesamt** kleiner als 30 ist. Wenn ja, werden die Blöcke im Körper durchlaufen.
3. Eine Zufallszahl im Bereich von 1 bis 6 wird erzeugt (um einen Würfelwurf zu simulieren) und in einer Variablen namens **gewürfelt** gespeichert.
4. Die gewürfelte Zahl wird ausgegeben.
5. Die Variable **insgesamt** wird um **gewürfelt** erhöht.
6. Wenn das Ende der Schleife erreicht ist, geht die Steuerung zurück zu Schritt 2.

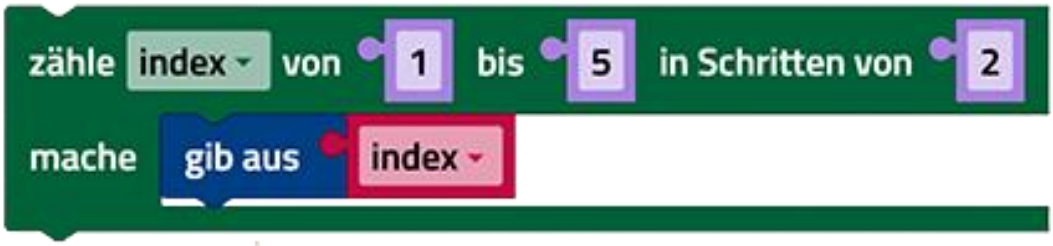
Nach Beendigung der Schleife werden alle nachfolgenden Blöcke (nicht dargestellt) durchlaufen. Im Beispiel endet der Schleifendurchlauf, nachdem eine gewisse Anzahl von Zufallszahlen im Bereich von 1 bis 6 ausgegeben wurde, und die Variable **insgesamt** hat dann als Wert die Summe dieser Zahlen, die mindestens 30 beträgt.

wiederhole-bis

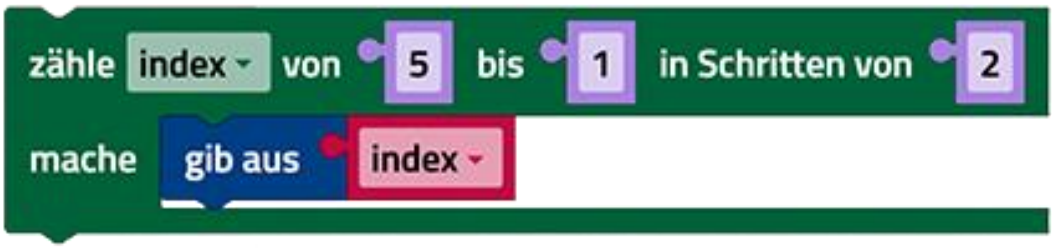
wiederhole solange-Schleifen wiederholen ihren Körper, **solange** eine Bedingung erfüllt ist. **wiederhole bis**-Schleifen sind ähnlich, mit dem Unterschied, dass sie ihren Körper so lange wiederholen, **bis** eine bestimmte Bedingung erfüllt ist. Die folgenden Blöcke sind äquivalent zum vorherigen Beispiel, weil die Schleife läuft, bis **insgesamt** größer oder gleich 30 ist.

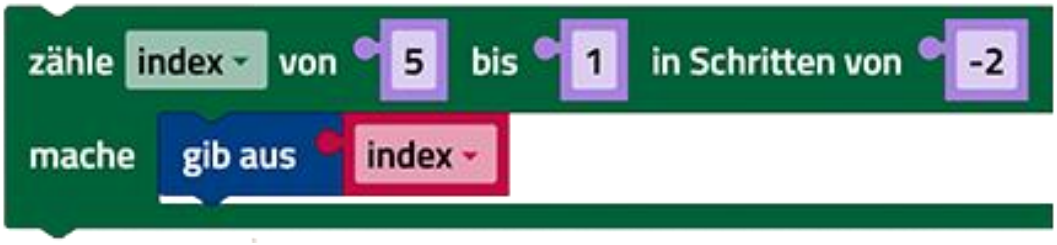
zählen-von-bis

Die **zählen-von-bis**-Schleife erhöht einer Variable den Wert, beginnend mit einem ersten Wert , endend mit einem zweiten Wert und in Schritten von einem dritten Wert, wobei der Körper für jeden Wert der Variable einmal ausgeführt wird. Das folgende Programm gibt zum Beispiel die Zahlen 1, 3 und 5 aus.



Wie die beiden folgenden Schleifen zeigen, die jeweils die Zahlen 5, 3 und 1 ausgeben, kann dieser erste Wert größer sein als der zweite. Das Verhalten ist das gleiche, egal ob der Inkrementbetrag (dritter Wert) positiv oder negativ ist.





für jeden

Der **für jeden**-Block ist ähnlich, wie die **zählen-von-bis**-Schleife, nur dass er statt der Schleifenvariable in einer numerischen Reihenfolge die Werte aus einer Liste der Reihe nach verwendet. Das folgende Programm gibt jedes Element der Liste "alpha", "beta", "gamma" aus:

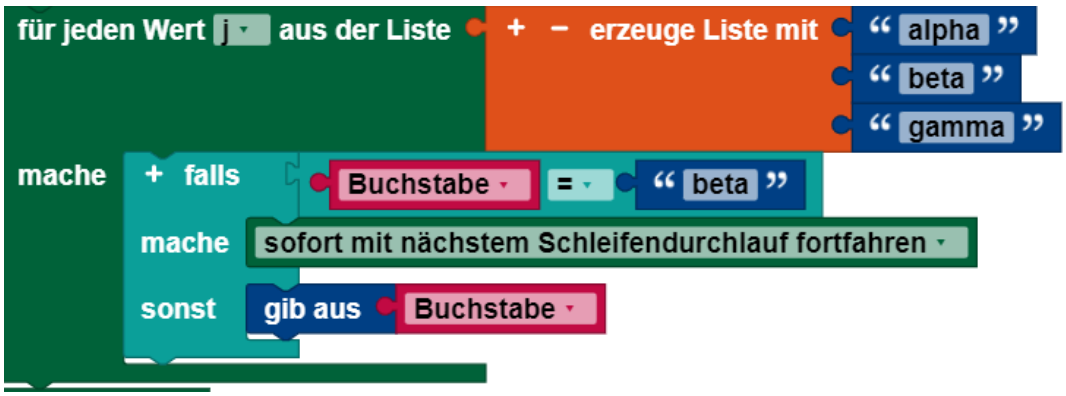
Schleifenabbruchblöcke

Die meisten Schleifen werden so lange durchlaufen, bis die Abbruchbedingung (bei **wiederhole**-Blöcken) erfüllt ist oder bis alle Werte der Schleifenvariable angenommen wurden (bei **zählen mit**- und **für jeden**-Schleifen). Zwei selten benötigte, aber gelegentlich nützliche Blöcke bieten zusätzliche Möglichkeiten zur Steuerung des Schleifenverhaltens. Sie können bei jeder Art von Schleife verwendet werden, auch wenn die folgenden Beispiel ihre Verwendung bei der **für jeden**-Schleife zeigen.

fahre-mit-nächster-Iteration-fort

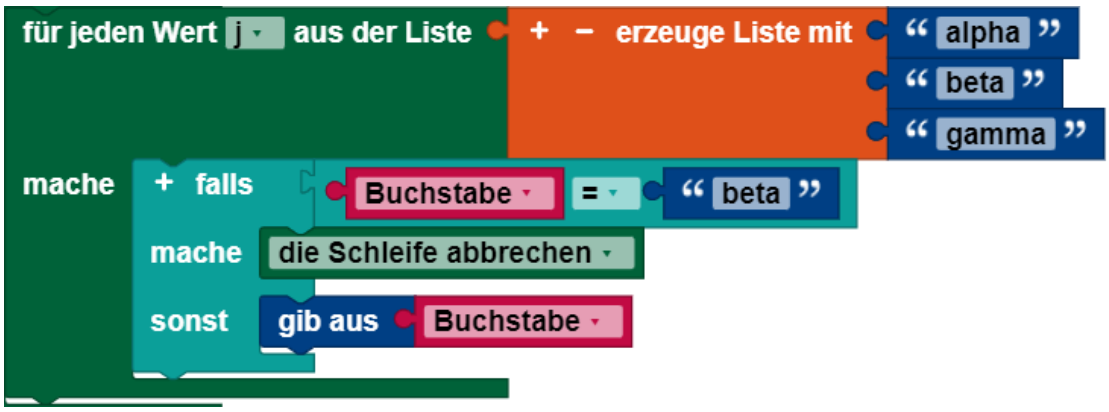
fahre-mit-nächster-Iteration-fort bewirkt, dass die restlichen Blöcke im Schleifenkörper übersprungen werden und die nächste Iteration der Schleife beginnt.

Das folgende Programm gibt bei der ersten Iteration der Schleife "alpha" aus. Bei der zweiten Iteration wird der Block **fahre mit nächster Integration fort** ausgeführt, wodurch die Ausgabe von "beta" übersprungen wird. Bei der letzten Iteration wird "gamma" gedruckt.



Schleifenabbruch

Der **schleifenabbruch**-Block ermöglicht einen vorzeitigen Ausstieg aus einer Schleife. Das folgende Programm gibt bei der ersten Iteration "alpha" und bricht bei der zweiten Iteration aus die Schleife ab, wenn die Schleifenvariable gleich "beta" ist. Der dritte Punkt in der Liste wird nie erreicht.



Mathematik

Die Blöcke, der Kategorie Mathematik werden genutzt, um Berechnungen anzustellen. Die Ergebnisse der Berechnungen können zum Beispiel als Werte für Variablen verwendet werden. Die meisten Mathematik-Blöcke beziehen sich auf allgemeine mathematische Berechnungen und sollten selbsterklärend sein.

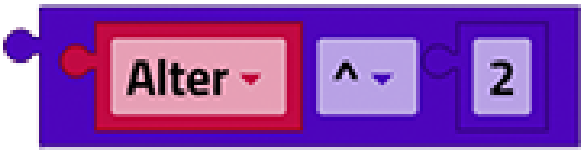
Blöcke

Zahlen

Nutze den Zahl-Block, um eine beliebige Zahl in dein Programm hinzufügen oder einer Variable diese Zahl als Wert zuzuweisen. Dieses Programm weist der Variabel **Alter** die Zahl 12 zu:

Einfache Rechnungen

Dieser Block hat die Struktur Wert-Operator-Wert. Als Operatoren stehen die Rechenarten +, -, ÷, × und ^ zu Verfügung. Der Operator kann über das Dropdown-Menü ausgewählt werden. Er kann unmittelbar auf Zahlen oder auch auf Werte von Variablen angewendet werden. Beispiel:



Dieser Block gibt Ergebnis 144 (12^2) aus.

Spezielle Rechnungen

Dieser Block wendet die, über das Dropdown-Menü ausgewählte Rechenart auf die dahinter platzierte Zahl oder auf den Wert der dahinter platzierten Variable an. Die zur Verfügung stehenden Rechenoperationen sind:

- Quadratwurzel,
- Betrag,
- natürlicher Logarithmus,
- dekadischer Logarithmus,
- Exponentialfunktion mit der Basis e (e^1, e^2,...),
- Exponentialfunktion mit der Basis 10 (10^1, 10^2,...),
- Vorzeichenwechsel (Multiplikation mit -1).

e ist hierbei die Euler'sche Zahl. Dieser Block zieht die Quadratwurzel aus 16 und setzt die Variable **i** auf das Ergebnis.

Trigonometrische Funktionen

Dieser Block funktioniert ähnlich wie der zuvor beschriebene Block, mit dem Unterschied, dass als Rechenoperationen die trigonometrischen Funktionen Sinus, Cosinus, Tangens und ihre Umkehrfunktionen genutzt werden. Die angegebene Zahl oder der Wert der angegebenen Variable wird also in die im Dropdown-Menü gewählte Funktion eingesetzt und das Ergebnis kann dann im Programm weiterverarbeitet werden. Zusätzlich gibt es noch den Block **arctan2 of X: ... Y: ...**, der es erlaubt, sich mit Hilfe von zwei reellen Zahlen (einzusetzen als X und Y) einen Funktionswert des arctan2 im Bereich von 360° ausgeben zu lassen.

Häufig verwendete Konstanten

Dieser Block funktioniert genauso wie der Zahl-Block, jedoch gibt man hier den Zahlenwert nicht selber an. Stattdessen sind häufig verwendete Konstanten (z.B. π) vorgespeichert. Die Konstante kann über das Dropdown-Menü ausgewählt werden.

Rest einer Division

Der **Rest von ...**-Block wird genutzt, um den Rest einer Division auszugeben. Dieses Programm weist der Variable **Rest** den Rest der Division von 3:2, also 1, zu:

Runden

Mit dem **runde** ...- Block lässt sich eine angegebene Dezimalzahlen oder der Wert einer angegebenen Variablen auf eine ganze Zahl runden. Dabei kann man im Dropdown-Menü drei Optionen wählen:

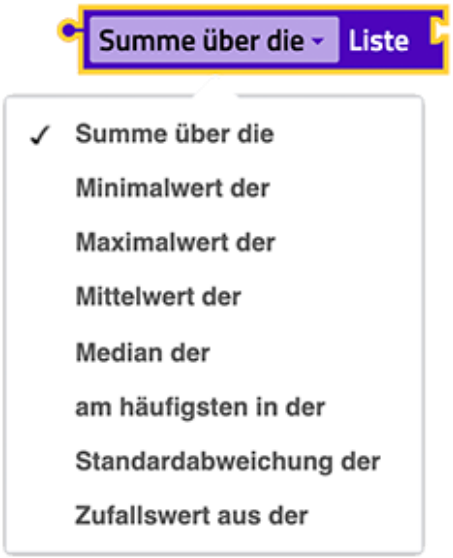
- mit "runde" kaufmännisch gerundet (z.B. wird 4,5 zu 5)
- mit "runde auf" wird aufgerundet (z.B. wird 5,1 zu 6)
- mit "runde ab" wird abgerundet (z.B. wird 5,9 zu 5).

Auswertung von Listen

Mit dem ... **der Liste**-Block kann man sich

- mit "Summe" die Summe aller Werte einer Liste,
- mit "min" den kleinsten Wert einer Liste,
- mit "max" den größten Wert einer Liste,
- mit "Mittelwert" den Mittelwert aller Werte einer Liste,
- mit "Median" den Median einer Liste,
- mit "Modalwert" den/die am häufigsten vorkommenden Wert einer Liste,
- mit "Standardabweichung" die Standardabweichung aller Werte einer Liste,
- mit "Zufallswert" einen zufälligen Wert aus einer Liste

ausgeben lassen. Alle diese Optionen können über das Dropdown-Menü des Blocks ausgewählt werden:



Eingabewerte einschränken

Der **beschränke ... von ... bis ...**-Block erlaubt es, Eingabewerte auf ein bestimmtes Intervall zu beschränken. Bevor ein Eingabewert weiterverarbeitet wird, wird getestet, ob er im festgelegten Intervall liegt. Es gibt drei Optionen, wie mit einem eingeben Wert verfahren wird:

- Der Wert liegt im Intervall, also wird er unverändert weitergegeben.
- Der Wert liegt unter der unteren Grenze des Intervalls, also wird diese untere Grenze weitergegeben.
- Der Wert liegt über der oberen Grenze des Intervalls, also wird diese obere Grenze weitergegeben.

In diesem Beispiel wird der Block genutzt, um den Wert der Variable **Geschwindigkeit** auf die vom Motor unterstützten Drehzahlen einzuschränken:

Zufällige Werte generieren

Die beiden Blöcke **zufällige Zahl von ... bis...** und **zufälliger Bruch** geben einen zufälligen Wert aus. Dabei gibt der **zufällige Zahl von ... bis... Block** eine Zahl aus dem definierten Intervall aus. Der Block **zufälliger Bruch** gibt hingegen einen Wert zwischen 0,0 (eingeschlossen) und 1,0(ausgeschlossen) aus.

Text

Beispiele für Texte sind:

"Ding		1"
"12.	März	2010"
""	(leerer	Text)

Text kann Buchstaben (klein oder groß geschrieben), Zahlen, Satzzeichen, andere Symbole und Leerzeichen enthalten.

Blöcke

Erstellung von Text

Der folgende Block erzeugt den Text "Hallo" und speichert ihn in der Variablen namens **Gruß:**

Der Block **erstelle Text aus** kombiniert den Wert der Variable **Gruß** und den neuen Text "Welt" zu dem Text "HalloWelt". Beachte, dass zwischen beiden Texten kein Leerzeichen steht, da in den beiden ursprünglichen Texten keines vorhanden war.

Um die Anzahl der Texteingaben zu erhöhen, klicke auf das (+) Symbol. Um die letzte Ausgabe zu entfernen, klicke auf das (-) Symbol.

Änderung von Text

Der Block **an ... anhängen** fügt den angegebenen Text an die angegebene Variable an. In diesem Beispiel ändert er den Wert der Variable **Gruß** von "Hallo" in "Hallo, da!":

Textlänge

Der **Länge von**-Block zählt die Anzahl der Zeichen (Buchstaben, Zahlen usw.), die in einem Text enthalten sind. Die Länge von "Wir sind #1!" ist 12, und die Länge des leeren Textes ist 0.

Prüfen auf leeren Text

Der Baustein **ist leer** prüft, ob der angegebene Text leer ist (die Länge 0 hat). Das Ergebnis ist im ersten Beispiel **wahr** und im zweiten Beispiel **falsch**.

Suchen von Text

Diese Blöcke können verwendet werden, um zu prüfen, ob ein Text in einem anderen Text vorkommt und wenn ja, wo er vorkommt. Zum Beispiel wird hier nach dem ersten Vorkommen von "a" in "Hallo" gefragt, das Ergebnis ist 2:

Dies fragt nach dem letzten Vorkommen von "a" in "Hallo", was ebenfalls 2 ergibt:

Unabhängig davon, ob das erste oder letzte Vorkommen ausgewählt wird, liefert dieser Block das Ergebnis 0, da "Hallo" kein "z" enthält.

Extrahieren von Text

Extrahieren eines einzelnen Zeichens

Dies ergibt "b", den zweiten Buchstaben in "abcde":

Dies liefert "d", den vorletzten Buchstaben in "abcde":

Dies liefert "a", den ersten Buchstaben in "abcde":

Dies erhält "e", den letzten Buchstaben in "abcde":

Dies erhält jeden der 5 Buchstaben in "abcde" mit gleicher Wahrscheinlichkeit:

Keiner von ihnen verändert den Text, aus dem extrahiert wird.

Extrahieren eines Textbereichs

Mit dem **im Text ... liefer Zeichenkette**-Block kann ein Textbereich extrahiert werden, der entweder mit:

- Buchstabe #
- Buchstabe # vom Ende
- erster Buchstabe

startet und mit:

- Buchstabe #
- Buchstabe # vom Ende
- letzter Buchstabe

endet.

Im folgenden Beispiel wird "abc" extrahiert:

Anpassen der Groß-/Kleinschreibung des Textes

Dieser Block erzeugt eine Version des Eingabetextes, die entweder in

- GROßSCHREIBUNG (alle Buchstaben in Großbuchstaben) oder
- kleinschreibung (alle Buchstaben sind Kleinbuchstaben) oder
- Substantive (erste Buchstaben Großbuchstaben, andere Buchstaben Kleinbuchstaben).

Das Ergebnis des folgenden Blocks ist "HALLO":

Nicht-alphabetische Zeichen sind davon nicht betroffen. Beachte, dass dieser Block auf Text in Sprachen ohne Groß- und Kleinschreibung, wie z. B. Chinesisch, nicht wirkt.

Trimmen (Entfernen) von Leerzeichen

Der folgende Block entfernt, je nachdem, was im Dropdown-Menü (kleines Dreieck) eingestellt wird, Leerzeichen:

- am Anfang des Textes
- am Ende des Textes
- an beiden Seiten des Textes

Das Ergebnis des folgenden Blocks ist "Hi du".

Leerzeichen in der Mitte des Textes sind nicht betroffen.

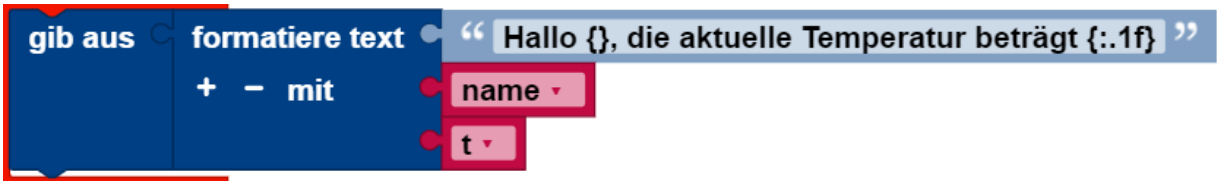
Text ausgeben

Der **gib aus**-Block bewirkt, dass der Eingabewert im Konsole-Fenster ausgegeben wird:

Auf keinen Fall wird er an den Drucker geschickt, wie der Name vielleicht vermuten lässt.

Text ausgeben mit Formatierung

Mit dem **formatiere text**-Block können Textausgaben mit Variableninhalt formatiert ausgegeben werden. Dabei werden alle Platzhalter {} im Text durch den Inhalt der nach dem Text angehängten Variablen ersetzt. In den geschweiften Klammern kann eine Formatierung angegeben werden. Die Formatierung **{:.1f}** gibt z.B. nur die erste Nachkommastelle der Kommazahl in der Variablen **t** aus.



Datenstrukturen

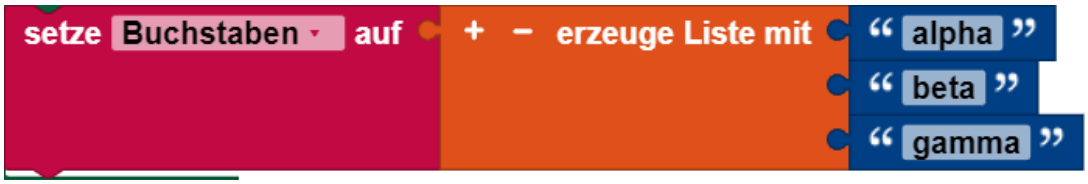
Listen

Wie in der Alltagssprache ist auch in ROBO Pro Coding eine Liste eine geordnete Sammlung von Elementen, wie z. B. eine "To-Do"-Liste oder eine Einkaufsliste. Elemente in einer Liste können von beliebigem Typ sein, und derselbe Wert kann mehrmals in einer Liste erscheinen.

Erstellen einer Liste

erstelle Liste mit

Mit dem Block **erstelle Liste mit** kann man die Anfangswerte in einer neuen Liste angeben. In diesem Beispiel wird eine Liste von Wörtern erstellt und in einer Variablen namens **Buchstaben** abgelegt:



Wir bezeichnen diese Liste als ("alpha", "beta", "gamma").

Dies zeigt die Erstellung einer Liste von **Zahlen**:

So wird eine Liste von **Farben** erstellt:

Es ist weniger üblich, aber möglich, eine Liste mit Werten unterschiedlichen Typs zu erstellen:

Anzahl der Eingänge ändern

Um die Anzahl der Eingänge zu ändern, klicke bzw. tippe auf das Zahnradsymbol. Dadurch wird ein neues Fenster geöffnet. Du kannst Elementunterblöcke von der linken Seite des Fensters in den Listenblock auf der rechten Seite ziehen, um einen neuen Eingang hinzuzufügen.:

Während das neue Element in diesem Beispiel unten hinzugefügt wurde, kann es überall hinzugefügt werden. In ähnlicher Weise können unerwünschte Elementunterblöcke aus dem Listenblock nach links gezogen werden.

Liste mit Element erstellen

Mit dem Block **erstelle Liste mit Element** kannst du eine Liste erstellen, die die angegebene Anzahl von Kopien eines Elements enthält. Die folgenden Blöcke setzen zum Beispiel die Variable **Wörter** auf die Liste ("sehr", "sehr", "sehr").

Prüfen der Länge einer Liste

ist leer

Der Wert eines **ist leer**-Blocks ist **wahr**, wenn seine Eingabe die leere Liste ist, und **falsch**, wenn es irgendetwas anderes ist. Ist diese Eingabe **wahr**? Der Wert des folgenden Blocks wäre **falsch**, weil die Variable Farben nicht leer ist: Sie hat drei Elemente.

Beachte die Ähnlichkeit mit dem **ist leer**-Block für Text.

Länge von

Der Wert des **Länge von**-Blocks ist die Anzahl der Elemente, die sich in der als Eingabe verwendeten Liste, befinden. Der Wert des folgenden Blocks wäre z. B. 3, da **Farbe** drei Elemente hat:

Beachte, dass der **Länge von**-Block angibt, wie viele Elemente in der Liste enthalten sind, und nicht, wie viele verschiedene Elemente in ihr enthalten sind. Zum Beispiel hat das Folgende den Wert 3, obwohl **Wörter** aus drei Kopien desselben Textes besteht:

Beachte die Ähnlichkeit mit dem Block **Länge von** für Text.

Suchen von Elementen in einer Liste

Diese Blöcke finden die Position eines Elements in einer Liste. Das folgende Beispiel hat den Wert 1, weil das erste Auftreten von "sehr" am Anfang der Wortliste steht (["sehr", "sehr", "sehr"]).

Das Ergebnis des Folgenden ist 3, weil das letzte Auftreten von "sehr" in **Wörter** an Position 3 ist.

Wenn das Element nirgendwo in der Liste vorkommt, ist das Ergebnis der Wert 0, wie in diesem Beispiel:

Diese Blöcke verhalten sich analog zu den Blöcken für das Finden von Buchstaben im Text.

Abrufen von Elementen aus einer Liste

Abrufen eines einzelnen Elements

Erinnere dich an die Definition der Liste **Farben**:

Der folgende Block erhält die Farbe Blau, weil es das zweite Element in der Liste ist (von links beginnend gezählt):

Dieser erhält Grün, weil es das zweite Element ist (vom rechten Ende aus gezählt):

Dieser erhält das erste Element, Rot:

Dies erhält das letzte Element, Gelb:

Dies wählt zufällig ein Element aus der Liste aus, wobei mit gleicher Wahrscheinlichkeit eines der Elemente Rot, Blau, Grün oder Gelb zurückgegeben wird.

Abrufen und Entfernen eines Elements

Mit dem Dropdown-Menü wird der Block **aus Liste ... abrufen** in den Block **aus Liste ... abrufen und entfernen** geändert, der die gleiche Ausgabe liefert, aber auch die Liste verändert:

Dieses Beispiel setzt die Variable **erster Buchstabe** auf "alpha" und lässt die restliche Buchstaben (["beta", "gamma"]) in der Liste.

Entfernen eines Eintrags

Wenn du im Dropdown-Menü **entfernen** wählst, verschwindet die Nase links vom Block:

Damit wird das erste Element aus **Buchstaben** entfernt.

Eine Subliste abrufen

Der Block **aus Liste ... Subliste abrufen** ähnelt dem Block in **aus Liste ... abrufen** mit dem Unterschied, dass er eine Subliste extrahiert und nicht ein einzelnes Element. Es gibt mehrere Optionen, den Anfang und das Ende der Subliste anzugeben:

In diesem Beispiel wird eine neue Liste **erster Buchstabe** erstellt. Diese neue Liste hat zwei Elemente: ("alpha", "beta").

Beachte, dass dieser Block die ursprüngliche Liste nicht verändert.

Hinzufügen von Elementen an eine Liste

Elemente in einer Liste ersetzen

Der Block **in Liste ... ersetze** ersetzt das Element an einer bestimmten Stelle einer Liste durch ein anderes Element.

Die Bedeutung der einzelnen Dropdown-Optionen findest du im vorherigen Abschnitt.

Das folgende Beispiel bewirkt zwei Dinge:

1. Die Liste **Wörter** wird mit 3 Elementen erstellt: ("sehr", "sehr", "sehr").
2. Das dritte Element in der Liste wird durch "gut" ersetzt. Der neue Wert von **Wörter** ist ("sehr", "sehr", "gut")

Elemente an einer bestimmten Stelle in eine Liste einfügen

Der **in Liste ... einfügen bei**-Block wird über das Dropdown-Menü des **in Liste ... ersetze**-Blocks aufgerufen:

Er fügt ein neues Element an der angegebenen Stelle in die Liste ein, und zwar vor dem Element, das sich zuvor an dieser Stelle befand. Das folgende Beispiel (das auf einem früheren Beispiel aufbaut) tut drei Dinge:

1. Die Liste **Wörter** wird mit 3 Elementen erstellt: ("sehr", "sehr", "sehr").
2. Das dritte Element in der Liste wird durch "gut" ersetzt. Der neue Wert von **Wörter** ist somit ("sehr", "sehr", "gut").

3. Das Wort "Sei" wird am Anfang der Liste eingefügt. Der endgültige Wert von **Wörter** ist somit ("Sei", "sehr", "sehr", "gut").

Zeichenketten aufteilen und Listen zusammenfügen

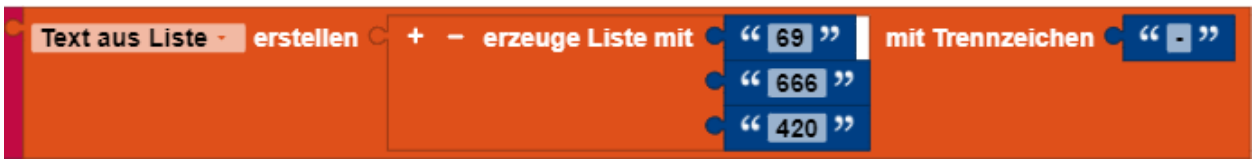
Eine Liste aus einem Text erstellen

Der Baustein **erstelle Liste aus Text** zerlegt den angegebenen Text mit Hilfe eines Begrenzungszeichens in Teile:

Im obigen Beispiel wird eine neue Liste zurückgegeben, die drei Textstücke enthält: "311", "555" und "2368".

Eine Text aus einer Liste erstellen

Der Baustein **erstelle Text aus Liste** fügt eine Liste mit Hilfe eines Trennzeichens zu einem einzigen Text zusammen:

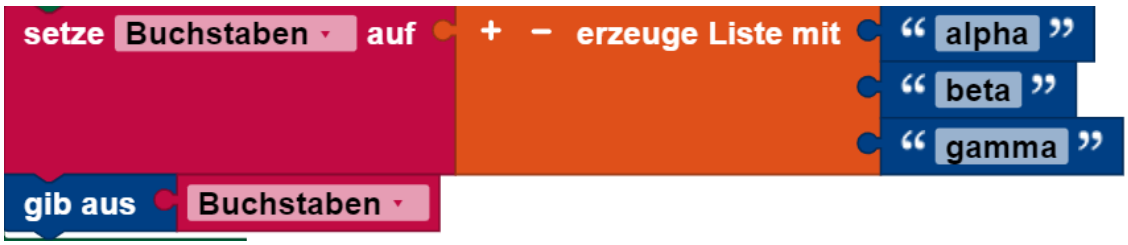


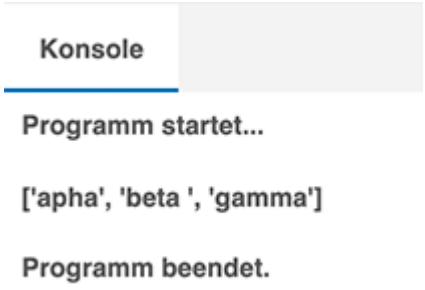
Im obigen Beispiel wird ein neuer Text mit dem Wert zurückgegeben: "311-555-2368".

Verwandte Blöcke

Drucken einer Liste

Der **drucken**-Baustein in der Kategorie Text kann Listen ausgeben. Das Ergebnis des folgenden Programms ist die abgebildete Konsolenausgabe:





Etwas für jedes Element in einer Liste durchführen

Der **für-jeden**-Block in der Kategorie Steuerung führt eine Operation für jedes Element in einer Liste aus. Dieser Block druckt zum Beispiel jedes Element in der Liste einzeln aus:

Dadurch werden die Elemente nicht aus der ursprünglichen Liste entfernt.

Siehe auch die Beispiele für die Schleifenabbruchblöcke.

Map

JSON

Benutzung

Die Kategorie Benutzung beinhaltet bei ROBO Pro Coding Blöcke folgender Art:

- Farbauswahl
- Warten
- Python Code

- Starten
- Funktionsausführung

Farbauswahl

Dieser Block dient als Eingabewert, wenn nach einer Farbe gefragt wird (z.B. beim Farbabgleich durch die Kamera). Durch Klicken bzw. Tippen auf die Farbe kann aus einer Farbpalette eine von 70 Farben ausgewählt werden.

Warten

Warten bis die Zeit abgelaufen ist

Der Block **warte () ...** hindert das Programm für die angegebene Wartezeit daran, weiterzulaufen. Dabei kann im Dropdown-Menü (kleines Dreieck) die Zeiteinheit und im Eingabefeld dahinter die gewünschte Länge der Pause gewählt werden.

Warten mit Bedingung

Beim **warte bis**-Block ist die Pause nicht an die Zeit sondern an die Erfüllung einer Bedingung (z.B. ob ein Taster gedrückt ist) geknüpft. Die Bedingung wird an den **warte bis**-Block angehängt.

Python-Code

Möchte man bestehenden Python-Code in ROBO Pro Coding integrieren, so kann man ihn in den **Python Code**-Block einfügen. Das Programm führt dann alles aus, was in dem Block in Python geschrieben wurde.

Starten

Auch der **starte wenn**-Block ist an eine Bedingung geknüpft. Erst wenn diese Bedingung erfüllt ist, startet das im Blockkörper stehende Programm.

Funktionsausführung

Mit dem **führe Funktion ... in einem Thread aus** lässt sich die ausgewählte Funktion in einem separaten Thread ausführen . Diese Maßnahme kann in manchen Fällen ermöglichen, dass ein Programm weiterhin auf Eingaben reagieren kann und schneller ausgeführt wird.

Variablen

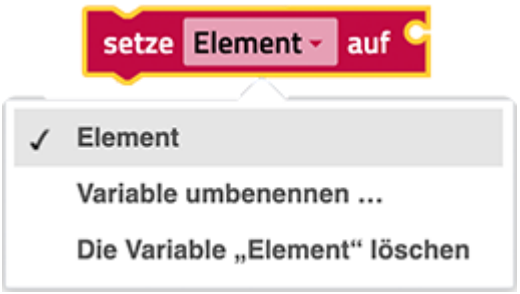
Wir verwenden den Begriff Variable so, wie er in der Mathematik und in anderen Programmiersprachen verwendet wird: ein benannter Wert, der

verändert (variiert) werden kann. Variablen können auf verschiedene Arten erstellt werden.

- Einige Blöcke wie **zähle mit** und **für jeden** verwenden eine Variable und definieren ihre Werte. Ein traditioneller Informatik-Begriff für solche Variablen lautet Schleifenvariablen.
- Benutzerdefinierte Funktionen (auch als "Prozeduren" bezeichnet) können Eingaben definieren, wodurch Variablen erzeugt werden, die nur innerhalb dieser Funktion verwendet werden können. Solche Variablen werden traditionell als "Parameter" oder "Argumente" bezeichnet.
- Benutzer können jederzeit Variablen über den **setze**-Block verändern. Diese werden traditionell als "globale Variablen" bezeichnet. Sie sind überall im Code von ROBO Pro Coding verwendbar.

Dropdown-Menü

Wenn du auf das Dropdown-Symbol (kleines Dreieck) einer Variable klickst, erscheint das folgende Menü:



Das Menü bietet die folgenden Optionen.

- die Anzeige der Namen aller vorhandenen, im Programm definierten Variablen.
- "Variable umbenennen...", d.h. die Änderung des Namens dieser Variable, wo immer sie im Programm erscheint (die Auswahl dieser Option öffnet eine Abfrage für den neuen Namen)
- "Variable löschen...", d.h. das Löschen aller Blöcke, die auf diese Variable verweisen, wo immer sie im Programm vorkommt.

Blöcke

Festlegen

Der **setze**-Block weist einer Variablen einen Wert zu und legt die Variable an, falls sie noch nicht existiert. Zum Beispiel wird so der Wert der Variable **Alter** auf 12 gesetzt:

Abrufen

Der **rufe ab**-Block liefert den in einer Variablen gespeicherten Wert, ohne ihn zu verändern:

Es ist möglich, aber eine schlechte Idee, ein Programm zu schreiben, in dem ein **rufe ab**-Block ohne einen entsprechenden vorherigen setze-Blockvorkommt.

Ändern

Der **ändere**-Block fügt eine Zahl zu einer Variablen hinzu.

Der **ändere**-Block ist eine Abkürzung für das folgende Konstrukt:

Beispiel

Betrachte den folgenden Beispielcode:

Die erste Reihe von Blöcken erzeugt eine Variable namens **Alter** und **setzt** ihren Anfangswert auf die Zahl 12. Die zweite Reihe von Blöcken **ruft** den den Wert 12 **ab**, addiert 1 dazu und speichert die Summe (13) in der Variablen. In der letzten Zeile wird die Meldung ausgegeben: "Herzlichen Glückwunsch! Du bist jetzt 13".

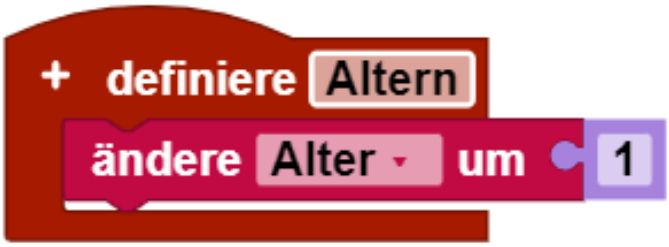
Funktionen

Funktionen dienen dazu, Teile des Codes wiederverwendbar zu machen und dadurch den Code insgesamt zu strukturieren. Füllt man einen Funktionsblock, so erscheint im Funktionen-Menü ein neuer Block, der den gleichen Namen trägt wie eben dieser Funktionsblock. Es ist nun möglich, in das Hauptprogramm nur noch den Block mit dem Namen der Funktion

einzusetzen. Wenn das Programm durchlaufen wird, leitet dieser Block zum Code in der gleichnamigen Funktion weiter und arbeitet diesen ab.

Einfache Funktion

Mit dem einfachen Funktionsblock lässt sich eine Funktion erstellen, die den im Textfeld eingegebenen Name trägt. Die Funktion kann beliebig viele Variablen enthalten, die über das Zahnradsymbol hinzugefügt werden können. Diese Funktion **Altern** addiert 1 zu der Variable **Alter**:



Die Funktion kann dann im Hauptprogramm genutzt werden:

Funktion mit Rückgabewert

Dieser Block erlaubt es eine Funktion mit Rückgabewert zu erstellen. Dieser Rückgabewert kann dann im Hauptprogramm weiterverwendet werden. Hier ein Beispiel:

Machine Learning

Die Gruppe "Machine Learning" enthält Blöcke zur Verwendung mit dem TensorFlow-Projekt und der USB-Kamera.

Erstellt eine Bildanalyse für ein Model im Pfad path auf dem TXT 4.0 Controller.

Erstellt eine Objekterkennung mit einem Standardmodell "Sortierstrecke mit KI".

Erstellt eine Objekterkennung für ein Model im Pfad path auf dem TXT 4.0 Controller.

Analysiert ein Bild mit einer Objekt- oder Bilderkennung. Ausgegeben werden die erkannten Eigenschaften, deren Wahrscheinlichkeit und bei einer Objekterkennung deren Position. Das Ergebnis dieses Blocks kann in eine Variable geschrieben werden, um dieses später im „get value of result item (item)" Block auszuwerten.

Gibt einen einzelnen Wert einer Eigenschaft des x-ten Ergebnisses einer Bild- oder Objektanalyse aus. Item kann dabei entweder direkt der „process image" Block oder dessen Ergebnisse aus einer Variable sein.

Importe

Importe enthält alle Funktionen aus selbst definierten Modulen in "lib".

Funktionen dienen dazu, Teile des Codes wieder verwendbar zu machen und dadurch den Code insgesamt zu strukturieren.

siehe "Funktionen"

Kommunikation

Sprachsteuerung

Blöcke für die die Kommunikation mit der App Voice Control.

wenn Befehl empfangen: Text

Wird ausgeführt, wenn ein neuer Text von App "Voice Control" empfangen wurde.

Text

Text ist der erkannte Sprachbefehl.

Cloud / MQTT

fischertechnik Cloud

Blöcke für die Kommunikation mit der fischertechnik Cloud.

Diese Blöcke werden für "Sensorstation IoT" und "Lernfabrik 4.0" verwendet.

MQTT Client

MQTT steht für "Message Queuing Telemetry Transport" Protokoll und wird oft bei IoT (Internet of Things) Anwendungen eingesetzt.

MQTT client create: websockets

Erstellt einen MQTT Client, mit dem Nachrichten empfangen und gesendet werden können. Es wird empfohlen die Ausgabe des Blocks in eine Variable zu schreiben, um den Client später in anderen Blöcken mehrfach verwenden zu können.

MQTT client ... connect

Verbindet einen MQTT Client mit einem MQTT Broker mit den angegebenen Einstellungen. Client kann der Block „MQTT client create" sein, oder dessen Ausgabe in einer Variablen. Host gibt die Adresse des MQTT Brokers an. Um den lokalen MQTT Broker zu verwenden, wird als Wert localhost oder 127.0.0.1 eingetragen. Um externe MQTT Broker zu verwenden, muss dessen IP-Adresse oder Hostname verwendet werden. Port gibt den Port an, an welchem der MQTT Broker verfügbar ist. Standard ist 1883 für MQTT Broker (fischertechnik Cloud) oder 2883 (GUI Applikation). Bei Verwendung externer MQTT Broker muss dessen Port eingetragen werden. Username und Password sind standardmäßig leer, bei externen Servern müssen dessen Anmeldeinformationen hier eingetragen werden.

MQTT client ... is connected

MQTT Client kann der Block „MQTT client create" sein, oder dessen Ausgabe in einer Variablen. Gibt „true" aus, wenn der angegebene MQTT Client mit einem MQTT Broker verbunden ist. Wenn der MQTT Client nicht verbunden ist, wird „false" zurückgegeben.

MQTT client ... disconnect

MQTT Client kann der Block „MQTT client create" sein, oder dessen Ausgabe in einer Variablen. Trennt den angegebenen MQTT Client vom MQTT Broker.

MQTT client ... publish

MQTT Client kann der Block „MQTT client create" sein, oder dessen Ausgabe in einer Variablen. Veröffentlicht mit dem angegebenen MQTT Client eine Nachricht „payload" in einen angegebenen Kanal „topic". Zusätzlich kann "qos" und "retain" angegeben werden, also ob neu verbundene Clients die Nachricht nach dem Senden empfangen sollen und mit welchem Sicherheitslevel die Nachricht gesendet werden soll.

MQTT client ... publish (mit Rückgabewert)

MQTT Client kann der Block „MQTT client create“ sein, oder dessen Ausgabe in einer Variablen. Veröffentlicht mit dem angegebenen MQTT Client eine Nachricht „payload“ in einen angegebenen Kanal „topic“. Zusätzlich kann "qos" und "retain" angegeben werden, also ob neu verbundene Clients die Nachricht nach dem Senden empfangen sollen und mit welchem Sicherheitslevel die Nachricht gesendet werden soll. Beim erfolgreichen Versenden wird „true“ ansonsten „false“ zurückgegeben.

MQTT client ... will set

MQTT Client kann der Block „MQTT client create“ sein, oder dessen Ausgabe in einer Variablen. Setzt die Nachricht „payload“, welche nach dem Trennen des MQTT Clients in einem angegebenen Kanal „topic“ versandt werden soll, und mit welchem Sicherheitslevel die Nachricht gesendet werden soll.

MQTT client ... subscribe

Abonniert mit einem MQTT Client einen Kanal. im Callback Argument wird die Funktion angegeben, welche beim Empfangen einer Nachricht ausgeführt werden soll. "Qos" gibt an, mit welchem Sicherheitslevel die Nachricht versendet werden soll.

subscribe callback ... : message

Definiert eine Funktion, welche durch Empfangen einer Nachricht ausgeführt werden soll. Message enthält die empfangene Nachricht.