micro:bit MicroPython Tutorial

Release 0.4

Harald March, Rae Harbird, Stephen Hailes

07.02.2024

Erste Schritte

Die Tutorials können auf <https://microbit-challenges-de.readthedocs.io/> abgerufen werden. Diese Dokumentation ist auch eingebunden in die Seite <https://micropython.matheharry.de/>, wo sie um eine Sammlung von interaktiven Quizfragen zur Wissensüberprüfung erweitert wurde, die den Lernprozess unterstützen sollen.

Sie basiert größtenteils auf der **englischsprachigen MicroPython** Dokumentation (es lohnt sich, auch dort vorbeizuschauen!) und der Arbeit von Rae Harbird.

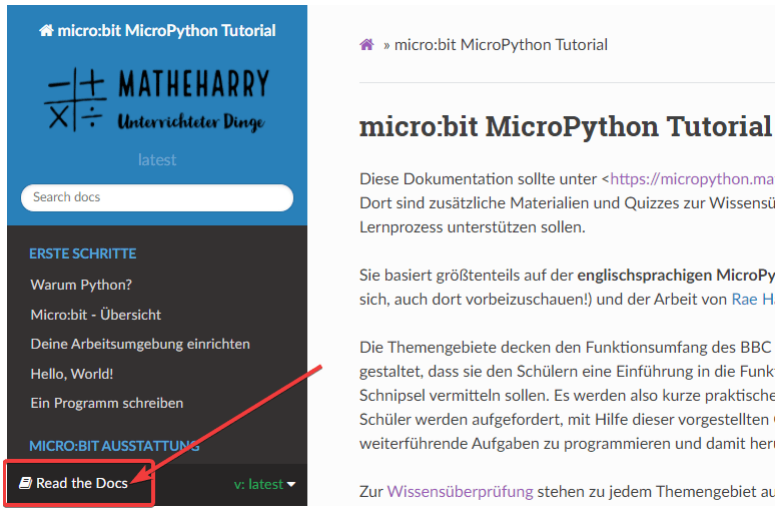
Die Themengebiete decken den Funktionsumfang des BBC micro:bit ab und sind so gestaltet, dass sie den Schülern eine Einführung in die Funktionen anhand kurzer Code-Schnipsel vermitteln sollen. Es werden also kurze praktische Beispiele gegeben und die Schüler werden aufgefordert, mit Hilfe dieser vorgestellten Grundbausteine Lösungen für weiterführende Aufgaben zu programmieren und damit herumzuexperimentieren.

Zur Wissensüberprüfung stehen zu jedem Themengebiet auch kleine Quizzes zu Verfügung.

Schülern das Programmieren mit einem Mikroprozessor wie dem micro:bit, der noch dazu mit Sensoren ausgestattet ist, beizubringen, ermöglicht es den Lernenden, sofortiges Feedback zu ihrem Code zu erhalten, ohne dass sie vorher etwas über Elektronik wissen müssen.

Die Challenge-Aufgaben können für Team-Wettbewerbe oder einfach zum Spaß im Klassenzimmer verwendet werden. Einige von ihnen basieren auf Übungen von M. Atkinson auf der großartigen Website Multiwingspan und wurden teilweise ein wenig angepasst.

Um diese Dokumentation im pdf-, epub- oder html-Format herunterzuladen, klicke auf den Link *Read the Docs* unten in der Seitenleiste auf der linken Seite:



Wenn du zu dieser Sammlung beitragen möchtest, nur zu! Installiere git und erstelle einen Branch. Es wäre toll, mehr Challenges und weitere Projekte zu haben.

KAPITEL 1

Warum Python?

Python ist eine sehr gute Wahl, egal ob du ein Anfänger bist, der die Grundlagen des Programmierens lernen möchte, oder ein erfahrener Programmierer, der eine große Anwendung entwickeln muss. Die Grundlagen von Python sind leicht zu erlernen und dennoch sind die Möglichkeiten enorm.

Python wurde in den späten 1980er Jahren von Guido van Rossum entwickelt und ist eine für Anfänger und Einsteige sehr gut geeignete Programmiersprache, die später auch den Fortgeschrittenen und Profis alles bietet, was man sich beim Programmieren wünscht.

1.1 Python ist einfach

Der Reiz von Python liegt in seiner Einfachheit und Schönheit. Für eine Programmiersprache ist Python relativ übersichtlich, und die Entwickler haben es bewusst so gehalten.

Eine grobe Einschätzung der Komplexität einer Sprache kann man anhand der Anzahl der Schlüsselwörter oder reservierten Wörter in der Sprache ablesen. Man versteht darunter Wörter, die vom Compiler oder Interpreter reserviert sind, weil sie eine bestimmte eingebaute Funktionalität der Sprache bezeichnen.

Python 3 hat 33 Schlüsselwörter und **Python 2** hat 31. Im Gegensatz dazu hat **C++** 62, **Java** hat 53 und **Visual Basic** hat mehr als 120.

Python Code hat eine einfache und saubere Struktur, die leicht zu erlernen und leicht zu lesen ist. In der Tat, wie du sehen wirst, erzwingt die Sprachdefinition selbst schon eine einfach zu lesende Codestruktur. So ermöglicht Python die Entwicklung von kompakten und lesbaren Programmen.

Programme, die in Python geschrieben sind, sind aus mehreren Gründen viel kürzer als vergleichbare Programme in zB. C, C++ oder Java:

- Die Datentypen erlauben es, komplexe Operationen in einer einzigen Anweisung auszudrücken;
- Anweisungen werden durch Einrückungen und nicht durch öffnende und schließende Klammern gruppiert;
- Variablen- oder Argumentdeklarationen sind nicht nötig.

Das alles sind Gründe, warum Python zB im PYPL - Index (PopularitY of Programming Language) an erster Stelle gereiht ist.

Abb. 1: Source: MaartenschrijftOorspronkelijk: Doc Searls op Flickr, CC BY-SA 2.0 <https://creativecommons.org/licenses/by-sa/2.0>, via Wikimedia Commons

KAPITEL 2

Micro:bit - Übersicht

Der BBC micro:bit ist ein Computer im Taschenformat, mit dem du lernen kannst, wie Software und Hardware zusammenarbeiten. Er hat eine LED-Anzeige, Tasten, Sensoren und viele Ein- und Ausgabefunktionen, die du programmieren und mit denen du herumexperimentieren kannst. Die neueste Version des micro:bit bietet zusätzlich die Möglichkeit, akustische Signale zu empfangen und wiederzugeben.

2.1 Ist der micro:bit ein Computer oder ein Mikrocontroller?

Vielleicht hast du gehört, dass der micro:bit auch als Mikrocontroller bezeichnet wird. Auf dem Board befindet sich nämlich ein Mikrocontroller, der so programmiert werden kann, dass er bestimmte Aufgaben ausführt, aber es hat auch Ein- und Ausgabegeräte wie Tasten und eine LED-Anzeige, was den miro:bit zu mehr als einem Mikrocontroller macht.

Ein Mikroprozessor plus Arbeitsspeicher, Datenspeicher und Ein- und Ausgabegeräte ergeben einen Computer (*EVA-Prinzip*). Der micro:bit ist lediglich ein Computer im Taschenformat, mit dem man alle möglichen Projekte erstellen kann: von Robotern bis hin zu Musikinstrumenten - die Möglichkeiten sind endlos.

Schauen wir uns die Funktionen, die du in deinen Programmen verwenden kannst, einmal an:

- 25 rote LED-Lichter, die Nachrichten und Bilder anzeigen können.
- Zwei programmierbare Tasten (A und B), mit denen du dem micro:bit sagen kannst, wann er etwas starten und stoppen soll.
- Ein Thermistor, um die Temperatur zu messen.
- Ein Lichtsensor, um die Veränderung des Lichts zu messen.
- Ein Beschleunigungssensor, um Bewegungen zu erkennen.
- Ein Magnetometer, um dir zu sagen, in welche Richtung du gehst.
- Eine Funk- und eine Bluetooth-Verbindung, um mit anderen Geräten zu interagieren.
- Ein Microphon und einen Lautsprecher
- Eine berührungssensitive Taste als Logo

Du kannst den micro:bit mit verschiedenen Programmiersprachen zum Leben erwecken.

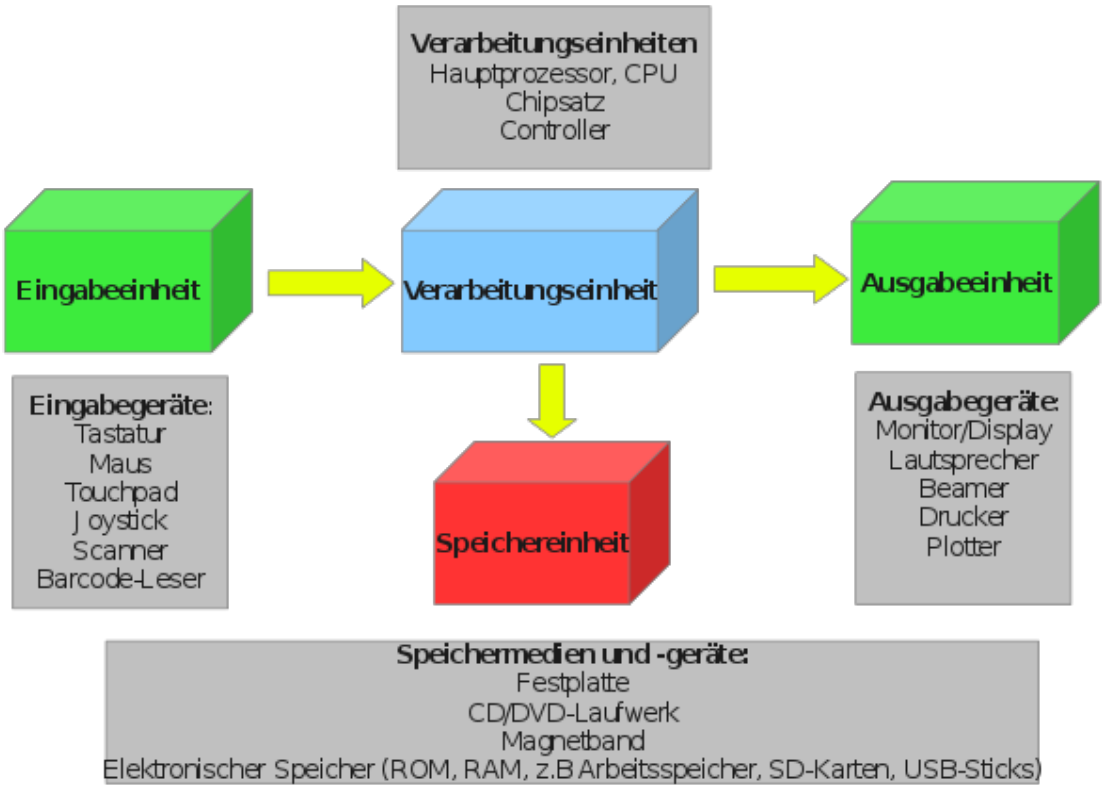


Abb. 1: Source: https://commons.wikimedia.org/wiki/File:EVA-Prinzip.svg

Abb. 2: Source: https://microbit.org/guide/features/

Zur Auswahl stehen **MicroPython, C++** oder **JavaScript**. Dieses Tutorial konzentriert sich auf das Programmieren von micro:bit mit MicroPython, aber wenn du bereits mit Python vertraut bist oder eine zusätzliche Herausforderung suchst, schau dir den Abschnitt Programmierung des micro:bit mit anderen Sprachen an.

MicroPython ist eine Version von Python, die auf Mikrocontrollern wie dem micro:bit laufen kann. Da die Funktionalität der beiden nahezu identisch ist (siehe hier für den Unterschied im Verhalten), beziehen wir uns in diesen Tutorials auf die verwendete Sprache als Python anstatt MicroPython.

Programmieren in Python besteht aus dem Beschreiben einer Reihe von Schritten, die ausgeführt werden sollen. An welche Regeln du dich dabei (ähnlich wie der Grammatik beim Sprachenlernen) ganz genau halten musst, lernst du Schritt für Schritt, wenn du deine ersten Programme schreibst.

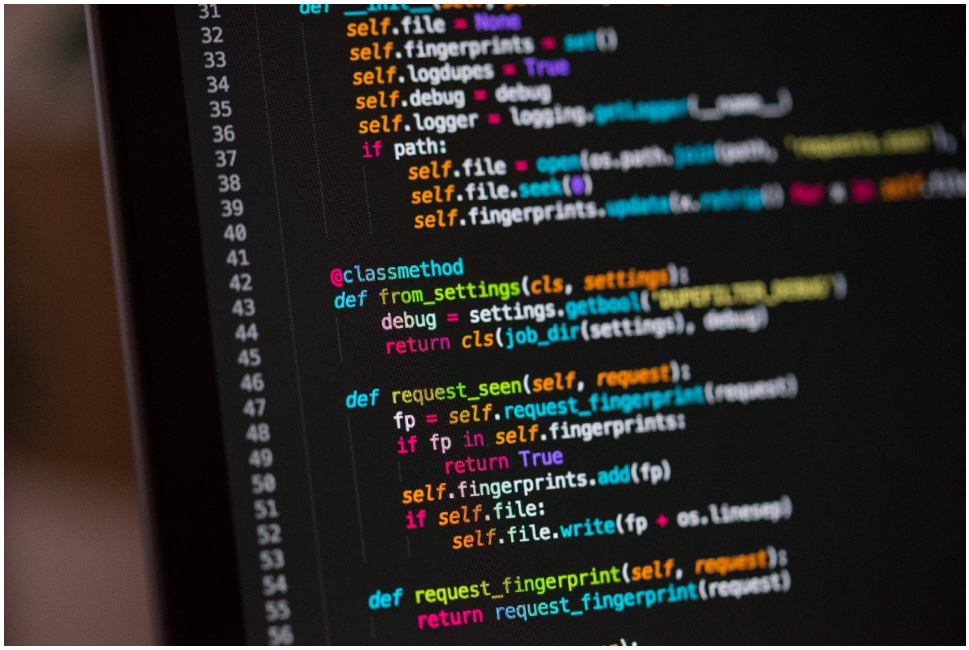


Abb. 3: Source: HOMEWORK

2.1.1 Leitfaden für diese Anleitung

Die Abschnitte dieses Tutorials sollen dir die Grundlagen der Programmierung und die Funktionen des micro:bit vermitteln. Das Ziel ist es, dich soweit mit dem Programmieren vertraut zu machen, dass du ein eigenes kleines Projekt erstellen kannst.

Unter anderem versuchen wir gemeinsam ein (immer besseres) *Schere-Stein-Papier-Spiel* zu machen - von ganz einfach bis … (ganz deiner Phantasie überlassen!). Das Erlernen der Programmiersprache Python sollte dabei irgendwie *nebenbei passieren* .

Du brauchst dich nicht unbedingt akribisch durch die ganze Theorie, die in Grundlagen der Programmierung behandelt wird, durcharbeiten, besonders wenn du ein Anfänger bist. Fange an, einfache Programme mit dem micro:bit zu schreiben und lerne nach und nach weitere Programmierkonzepte kennen indem du weiterliest und mit dem Code herumexperimentierst.

Wenn du schon mit der grafischen Programmieroberfläche MakeCode vertraut bist, lass dir deine Programme auch einmal als Python-Code darstellen. Allerdings unterscheidet sich der Code dort etwas von den MicroPython-Befehlen, die wir hier verwenden. Auch wenn die Syntax (so nennt man die Regeln und den Wortschatz, die für eine Sprache gelten) unterschiedlich ist, kann man so doch viel über den Aufbau und Ablauf eines Python-Programms lernen.

Fühle dich frei, die Teile zu überspringen, in denen du dich sicher fühlst und wähle die Teile, die dir wichtig sind. Während du mehr über das Programmieren lernst, wirst du natürlich immer bessere und effizientere Wege finden, um deine Projekte der Vergangenheit zu erledigen, aber im Moment solltest du dich darauf konzentrieren, den Einstieg zu finden.

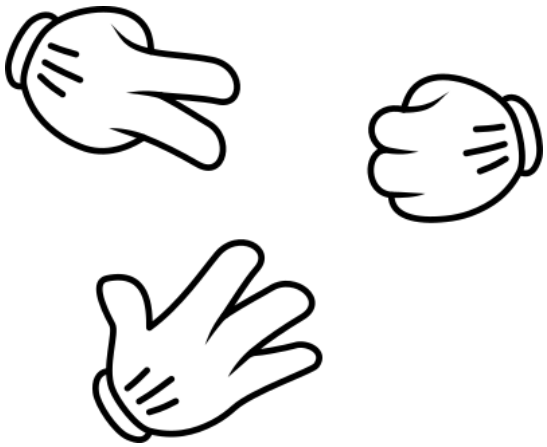


Abb. 4: Source: https://openclipart.org/detail/213382/rockpaperscissors

Bemerkung: Wenn du dir beim Lesen der Tutorials schwer tust und dich nicht auskennst oder du das Gefühl hast, dass du mehr Anleitung brauchst, um mit dem Programmieren beginnen zu können, lass dich nicht entmutigen!

Es gibt nichts Gutes außer man tut es! Dieser Satz von Erich Kästner beschreibt perfekt das Programmierenlernen. Man müsste ihn vielleicht noch ein bisschen erweitern mit: **… und dann tut man es noch einmal, und wenn's immer noch nicht funktioniert eben noch einmal und noch einmal!**

Die hier behandelten Themen werden in der Regel nur gestreift und viele Dinge werden absichtlich nicht erklärt, weil eigentlich alles im Internet zu finden ist. Hier sollst du die Grundlagen kennenlernen und vor allem ermutigt werden, Dinge einfach auszuprobieren! Einige der wichtigen Fähigkeiten die du im Laufe der Schule und deiner späteren Arbeit brauchen wirst, sind das selbstständige Arbeiten mit Unterlagen und die Fähigkeit, offizielle Dokumente zu lesen. Daher solltest du ruhig auch andere (und bessere) Unterlagen lesen, als die, die du hier findest.

Es gibt eine Reihe von kostenlosen Online-Kursen, die dir die Grundlagen der Programmierung mit Python näher

bringen, wie zum Beispiel python-lernen.de. Dringend zu empfehlen ist auch dieses YouTube-Tutorial, das dir die „Sprache“ Python und ihre, wie bei allen Programmiersprachen ganz streng einzuhaltende, Grammatik erklärt.

Beginne auf jeden Fall damit, die ersten paar Lektionen Schritt für Schritt nachzuvollziehen, schau an den oben erwähnten Stellen nach, wie Python funktioniert, wenn etwas nicht so hinhaut wie du dir das gedacht hast, und betrachte jeden Fehler als das was er ist - als einen wichtigen Schritt zum Meistercoder!

Denn mit der Zeit wird alles einen Sinn ergeben.

Wenn deine Fähigkeiten mittel bis fortgeschritten sind, wirst du diese Dokumentation vielleicht nicht sehr interessant finden. Wie auch immer, der micro:bit ist ein äußerst flexibles Gerät und du könntest dann vielleicht die micro:bit runtime erkunden, die dir mehr Flexibilität bei der Verwendung des Geräts bietet.

KAPITEL 3

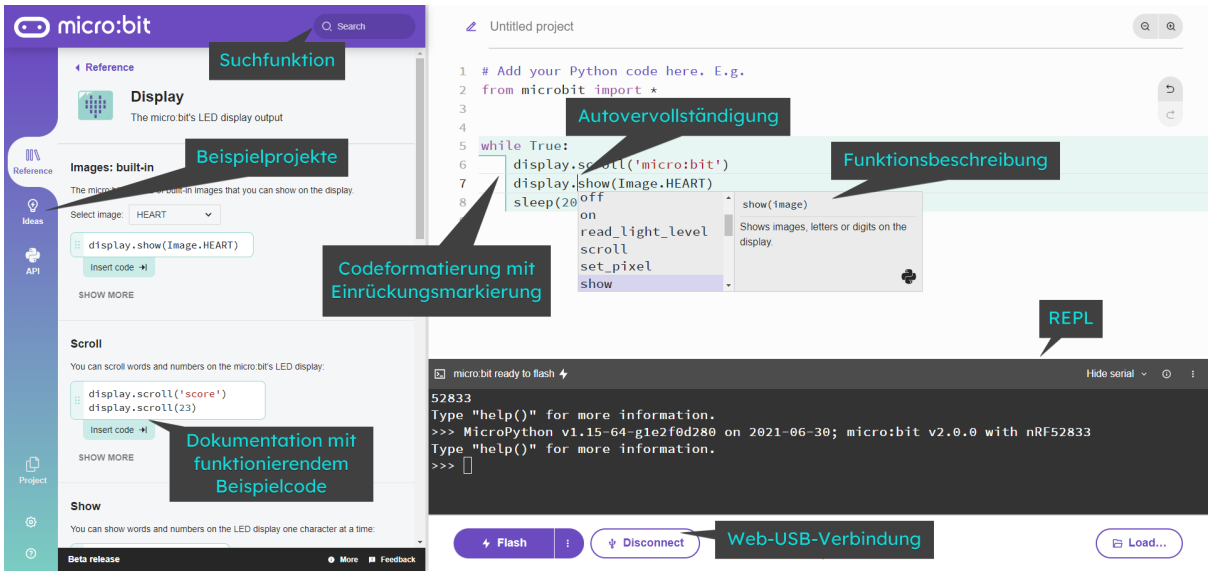
Deine Arbeitsumgebung einrichten

Bevor du mit dem Programmieren beginnst, benötigst du einen Quellcode-Editor, um Programme für den micro:bit schreiben, laden und ausführen zu können. Es gibt im Wesentlichen drei Optionen:

- den micro:bit Web-Editor (wer am neuesten Stand sein will kann auch die Beta-Version verwenden!)
- den Mu Editor am PC (vor allem wenn man sich über den micro:bit hinaus mit Python beschäftigen möchte!)
- die MicroPython App am Chromebook

3.1 Web Editor

Der neue browserbasierter Code-Editor wurde von Grund auf mit Blick auf den Einsatz im Unterricht entwickelt. Er soll den Zugang zum textbasierten Programmieren auch für Schülerinnen und Schüler, die mit dem Thema Coding wenig anzufangen wissen, leichter machen, indem **bekannte Lernbarrieren beseitigt** wurden. Er bietet bei jedem Schritt auf den micro:bit zugeschnittene Hilfen an und ist für unterschiedlichste Jahrgänge perfekt geeignet.



Um den Editor zu verwenden, den Link unbedingt in einem neuen Tab öffnen (per Rechtsklick).

3.1.1 Funktionierende Code-Beispiele und Code-Referenz durchsuchen

Wenn du mit der textbasierten Programmierung beginnst, ist es oft schwer zu wissen, was genau du eingeben musst.

Im Abschnitt „Referenz“ findest du Beispiele für funktionierenden Code, den du per Drag & Drop in den Editor ziehen und sofort verwenden kannst. Das fördert unabhängiges und kreatives Lernen, indem es das Entdecken und Anwenden von micro:bit-Funktionen und grundlegenden Computerkonzepten wie Schleifen, Variablen und Entscheidungsstrukturen erleichtert.

Diese Auswahl an Codeschnipseln und eine Reihe von vorgefertigten Bildern, Sounds und Musik sorgen so von Anfang an für Abwechslung und Spaß bei der Programmierung.

3.1.2 Intelligente Autovervollständigung

Die Autovervollständigungsfunktion zeigt dir während der Eingabe gültige Optionen an, damit du dich nicht an die genaue Syntax jedes Befehls erinnern musst. Die in einem DropDown-Menü erscheinden Vorschläge können mittels ENTER-Taste übernommen werden.

3.1.3 Fehlererkennung

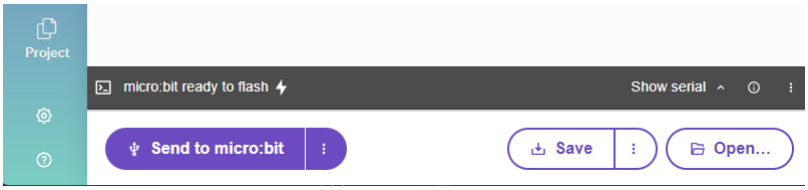
Der Editor zeigt auch potenzielle Fehler an, z. B. Variablen oder Funktionen, die nicht definiert wurden, oder Code, der nicht erreichbar ist. Das kann helfen, einfache Tippfehler zu erkennen, bevor du den Code auf einen micro:bit überträgst.

Normalerweise werden Fehler in Python erst bei der Ausführung erkannt und angezeigt, weshalb diese Funktion einen Vorteil zum Mu-Editor darstellt.

3.1.4 Codeformatierung

Einrückungen spielen in Python eine zentrale Rolle und können am Anfang leicht zu Fehlern bei der Eingabe führen. Deshalb verfügt der Webeditor über eine Linien- und Farbhervorhebung, die dir zeigt, wo und wie der Code eingerückt werden sollte, damit du deinen Code leichter korrigieren kannst.

3.1.5 Menü



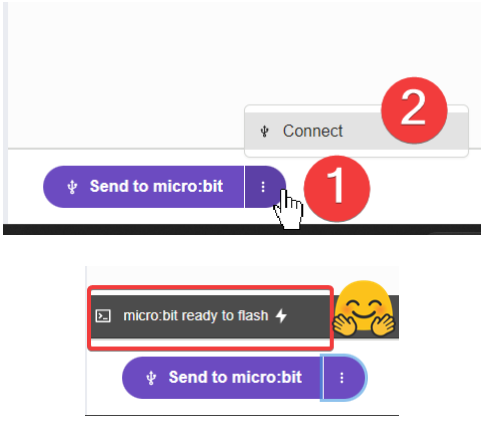
Das Hauptmenü des Editors enthält verschiedene Buttons, um deinen Code auf dem micro:bit ausführen zu lassen.

Zuerst musst du den micro:bit mit dem Computer und dem Editor mittels USB-Kabel verbinden. Du findest die Verbindungsschaltfläche, wenn du die 3 Punkte neben dem `Send to micro:bit` - Button anklickst und `Connect` auswählst.

Du wirst dann Schritt für Schritt durch den Verbindungsprozess geführt - beachte die sich öffnenden Fenster mit der Anleitung!

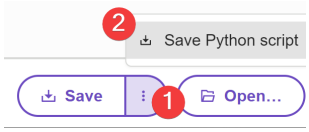
Wenn alles funktioniert hat siehst du folgendes Menü:

Schreibe dein Skript im Editorfenster und klicke auf den `Send to micro:bit` Button, um es direkt auf den micro:bit zu übertragen.



Wenn das nicht funktioniert, stelle sicher, dass dein micro:bit als USB-Speichergerät in deinem Dateisystem-Explorer aufscheint.

Klicke auf den `Save` Button, um deine „hex"-Datei auf dem Computer zu speichern. Über das 3-Punkte-Menü kann auch der Python-Code gespeichert werden.



Mittels `Open...` Button kann eine hex- oder Python-Datei hochgeladen und geöffnet werden.

3.1.6 REPL-Befehlszeile

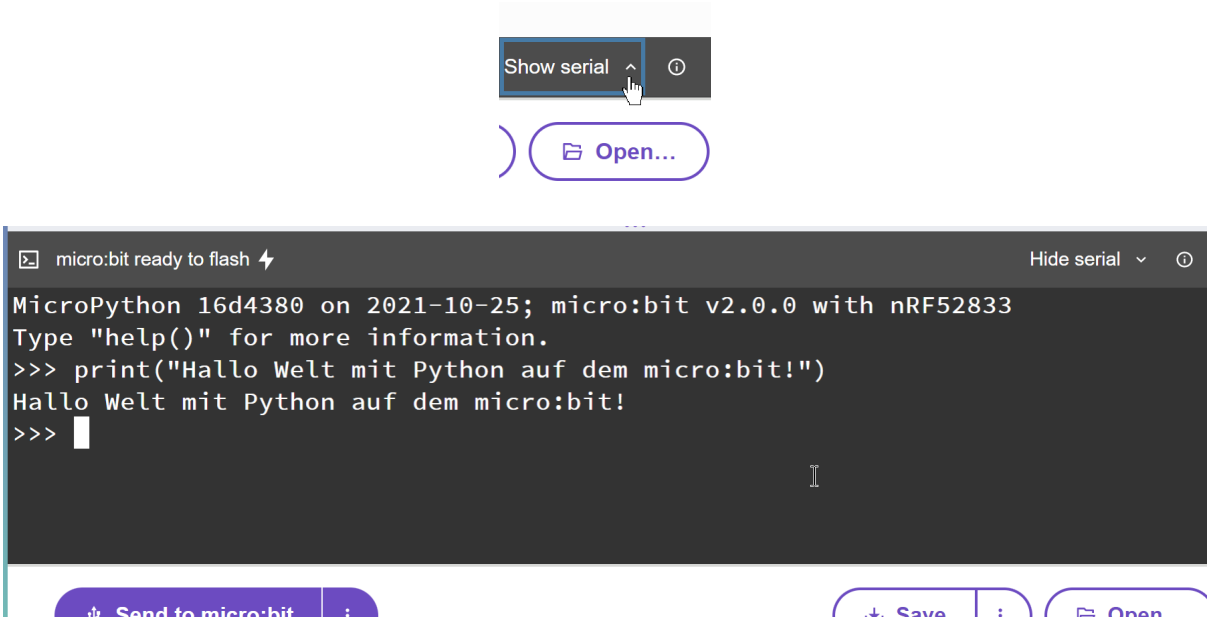
Der Webeditor erlaubt die Verwendung einer REPL-Konsole über den `Show Serial` Button unterhalb des Codes, um Befehle direkt auf dem micro:bit ausführen zukönnen. Es gibt auch in der rechten Leiste einen `Show Serial` Button, aber dieser ist nur Teil der Vorschau-Simulation!

Dazu wird die sogenannte REPL - Konsole (Read, Evaluate, Print Loop) gestartet, die es dir erlaubt, direkt auf dem micro:bit mit MicroPython zu arbeiten. Dort kannst du deine Befehle der Reihe nach eingeben und sie werden sofort ausgeführt.

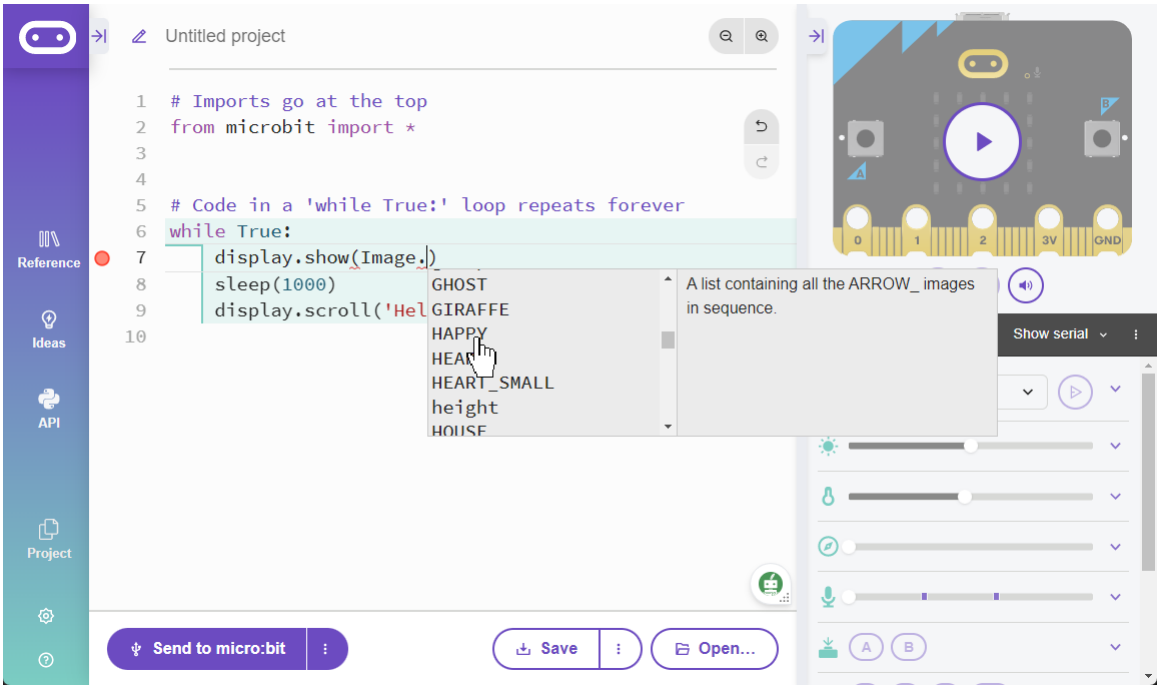
Wie der Name sagt, handelt es sich dabei um eine dauerhaft ausgeführte Schleife, die folgendermaßen abläuft, um den auf der Kommandoze eingegebenen Python-Code auszuführen:

- lesen (**R** ead): Lies die Benutzereingabe
- auswerten (**E** valuate): Überprüfe den Code und führe ihn aus
- ausgeben (**P** rint): Gib das Ergebnis aus
- Schleife ausführen (**L** oop): Gehe in der Schleife zurück zum Anfang

So wird es dir ganz einfach gemacht, Befehle auszuprobieren und herumzuexperimentieren! Außerdem werden Fehlermeldungen auf der REPL-Konsole ausgegeben und erleichtern so das Debuggen (so nennt man die Fehlersuche).



3.1.7 Text-Editor



Der Texteditor versucht zu helfen, indem er den Text einfärbt, um zu zeigen, was die verschiedenen Teile des Programms sind. Zum Beispiel sind die Python-Schlüsselwörter (Wörter, die in die Python-Sprache eingebaut sind) lila. Konstante Werte werden grün dargestellt und rote Schrift stellt Zeichenketten (Strings) dar.

Alle Zeilen sind nummeriert, wobei die aktuelle Zeile hervorgehoben ist.

Zusammengehörige, eingerückte Blöcke werden außerdem markiert, was die Struktur des Codes herausstreicht und gerade in Python eine wichtige Rolle beim Auffinden von Fehlern spielt.

Bemerkung: Für diejenigen, die vorher mit Python gearbeitet haben: MicroPython unterstützt keine regulären externen Python-Bibliotheken, da viele zu groß für ein Embedded Gerät sind. Allerdings wurde ein Subset speziell für die MicroPython-Umgebung neu erstellt.

3.2 Mu-Editor

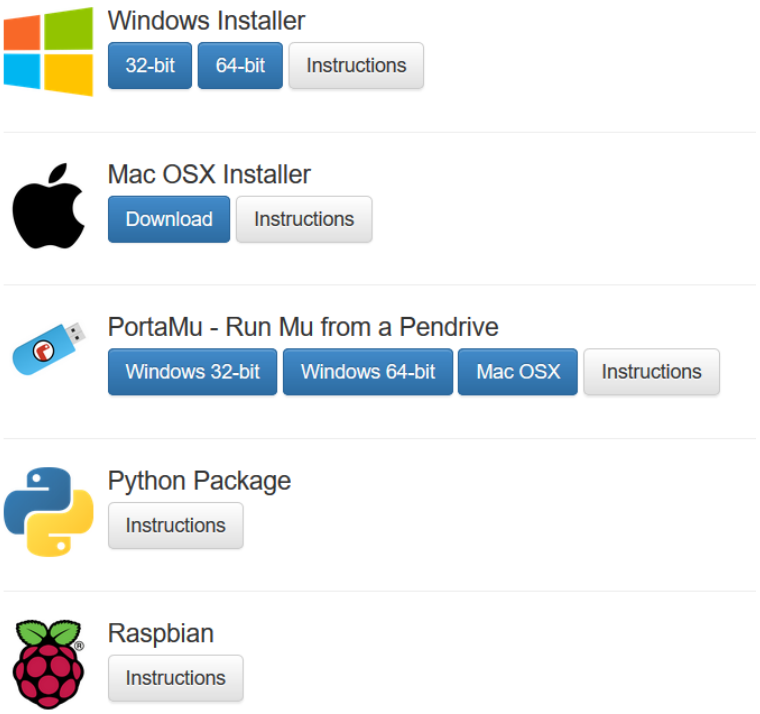
Der Editor Mu richtet sich speziell an Programmieranfänger und war bis vor kurzem noch die komfortabelste Möglichkeit, Pythonprogramme für den micro:bit zu erstellen. Inzwischen ist der Web-Editor schon so ausgereift, dass Mu eigentlich nicht mehr unbedingt benötigt wird.

Da Mu aber über die micro:bit-Programmierung weit hinausgeht und sogar die Entwicklung einfacher Computerspiele mittels **PyGame** unterstützt, soll er hier als weiterführende Alternative wärmstens empfohlen werden.

Um Mu herunterzuladen, gehe auf die Mu Website. Hier steht eine Schritt-für-Schritt-Anleitung zur Verfügung.

Du kannst verschiedene Optionen wählen, um Mu zu installieren. Die, die du höchstwahrscheinlich auf deinem eigenen Gerät benutzen wirst, wenn du Administratorenrechte hast, ist ein Installer für dein Gerät (Mac/Windows), oder die Installation durch ein Python-Paket (pip) über die Kommandozeile, wenn du Python schon auf deinem Computer installiert hast.

Falls du keine Administratorenrechte besitzt und Python auf deinem Gerät nicht installiert ist, bietet sich die portable Version *PortaMu* an. Diese muss man einfach in einen Ordner entpacken und hat damit sofort eine anfängerfreundliche Programmierumgebung zur Verfügung, die ohne Installation auf allen Geräten funktioniert. (Momentan ist leider **keine portable Version verfügbar!**)



Sobald der Editor installiert ist, starte ihn und schließe den micro:bit an deinen Computer an. Mu erkennt ihn automatisch und du kannst sofort loslegen.

3.2.1 REPL

Der Button **REPL** (Read, Evaluate, Print Loop) erlaubt es dir, dynamisch mit MicroPython auf dem micro:bit zu arbeiten, indem du die REPL-Befehlszeile direkt auf deinem micro:bit benutzt und Befehle der Reihe nach eingibst.

1. LESEN (read): Lies die Benutzereingabe
2. AUSWERTEN (evaluate): Überprüfe den Code und führe ihn aus
3. AUSGEBEN (print): Gib das Ergebnis aus
4. SCHLEIFE (loop): Gehe in der Schleife zurück zu Schritt 1

So wird es dir ganz einfach gemacht, Befehle auszuprobieren und herumzuexperimentieren! Außerdem werden Fehlermeldungen auf der REPL-Konsole ausgegeben und erleichtern so das Debuggen (so nennt man die Fehlersuche)

KAPITEL 4

Hello, World!

Der traditionelle Weg, mit der Programmierung in einer neuen Sprache zu beginnen, ist, den Computer dazu zu bringen, „Hello, World!“ zu sagen.

Mit MicroPython ist das ganz einfach:

```
from microbit import *
display.scroll("Hello, World!")
```

Sehen wir uns diese 2 Zeilen Code nun etwas genauer an.

4.1 Erklärungen und Grundbegriffe

Jede Zeile führt etwas Bestimmtes aus. Die erste Zeile:

```
from microbit import *
```

sagt MicroPython, dass es sich alles besorgen soll, was es benötigt, um mit dem BBC micro:bit zu arbeiten. Alle Werkzeuge dazu befinden sich in einem Modul namens `microbit` (ein Modul ist eine Bibliothek mit bereits existierendem Code).

Wenn du etwas mit `import` importierst, sagst du MicroPython, dass du es benutzen willst, und `*` ist Pythons Art, *„alles“* zu sagen. Also bedeutet `from microbit import *` auf Deutsch:

„Ich möchte alles aus der Code-Bibliothek *microbit* verwenden können“.

Die zweite Zeile:

```
display.scroll("Hello, World!")
```

weist MicroPython an, das LED-Display zum Scrollen der Zeichenkette „Hello, World!“ zu verwenden.

Der `display` Teil dieser Zeile ist ein sogenanntes **Objekt** aus dem `microbit`-Modul, das das LED-Display des Gerätes repräsentiert. Wir sagen ganz allgemein „Objekt“ zu allen möglichen Bestandteilen des microbits, die mit unserem Code zusammenarbeiten sollen. Genaugenommen ist in Python alles ein Objekt. Man redet deshalb auch von *objektorientierten Programmiersprachen*.

Wir können dem LED-Display oder irgendeinem anderen Objekt sagen was es tun soll, indem wir zuerst einen Punkt ans Ende seines Namens setzen: `display.`

Gleich nach dem Punkt folgt dann etwas, das wie ein Befehl aussieht (tatsächlich nennt man das eine **Methode**). In diesem Fall benutzen wir die Methode `scroll` die eine Laufschrift erzeugt: `display.scroll()`

Da `scroll` wissen muss, welche Zeichen über das Display „gescrollt“ werden sollen, übergeben wir diese Zeichen zwischen doppelten Anführungszeichen (`"`) innerhalb von Klammern (`(` und `)`). Diese werden die **Argumente** genannt.

Zum Beispiel bedeutet `display.scroll("Hello, World!")` auf Deutsch: „Ich möchte, dass du den Text ‚Hello, World¡ über den Bildschirm laufen lässt“.

Wenn eine Methode keine Argumente benötigt, heißt das, dass die Klammern leer bleiben. Das wäre zB bei der Methode `display.clear()` zum Löschen des Bildschirms der Fall. Die Klammern dürfen also nie weggelassen werden!

Kopiere den „Hello, World!“-Code in deinen Editor und flashe ihn auf das Gerät.

- Kannst du herausfinden, wie du die Nachricht ändern kannst?
- **Kannst du den micro:bit dazu bringen, dass er „Hallo“ zu dir sagt? Er könnte zum Beispiel „Hallo, Ladislaus!“ sagen.**
  Kleiner Tipp: du musst das Argument der Scroll-Methode ändern.

Warnung: Es kann sehr leicht sein, dass etwas nicht funktioniert. :-)

Dann beginnt der Spaß erst richtig! MicroPython versucht, hilfreich zu sein. Wenn es auf einen Fehler stößt, scrollt es eine hilfreiche Nachricht auf dem micro:bit Display. Wenn möglich, wird es dir die Zeilennummer sagen, in der der Fehler gefunden werden kann.

Python erwartet, dass du **EXAKT** das Richtige tippst. Also, zum Beispiel, `Microbit`, `microbit` und `microBit` sind alles unterschiedliche Begriffe für Python. Wenn MicroPython sich über einen `NameError` beschwert, passiert das wahrscheinlich deshalb, weil du etwas ungenau getippt hast. Es ist wie der Unterschied zwischen „Marc“ und „Mark“. Es sind zwei völlig verschiedene Personen, obwohl sich ihre Namen sehr ähnlich sehen.

Wenn MicroPython sich über einen `SyntaxError` beschwert, hast du den Code einfach auf eine Weise eingegeben, die MicroPython nicht verstehen kann. Überprüfe, ob du nicht irgendwelche Sonderzeichen wie `"` oder `:` vergessen oder hinzugefügt hast. Das ist so, als ob du einen Punkt in der Mitte eines Satzes stehen hast. Es ist dann schwer zu verstehen, was du genau meinst.

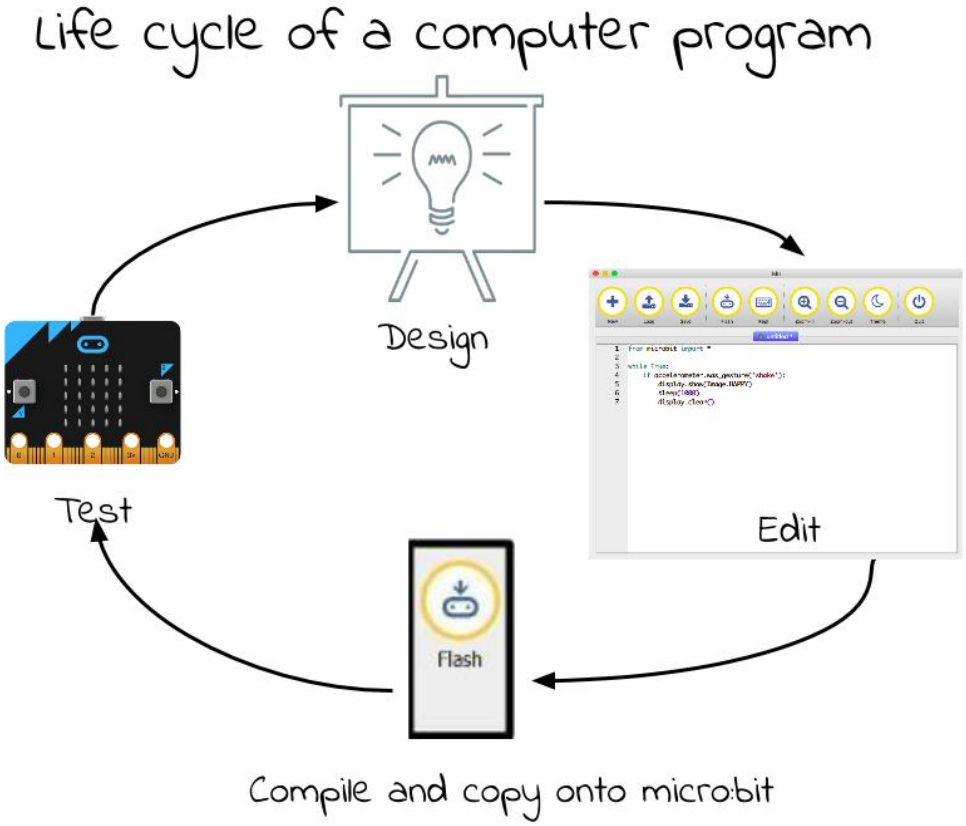
Wenn dein Microbit nicht mehr reagiert:

Wenn du keinen neuen Code flashen oder keine Befehle in der REPL eingeben kannst, versuche es mit einem Neustart. Das heißt, ziehe das USB-Kabel ab (und das Batteriekabel, falls es angeschlossen ist) und stecke danach das Kabel wieder ein. Eventuell musst du auch deine Code-Editor Anwendung beenden und neu starten.

KAPITEL 5

Ein Programm schreiben

Im Allgemeinen besteht der Prozess des Code-Entwurfs aus diesen 4 Schritten. Du kannst davon ausgehen, dass du die Schleife ein paar Mal durchlaufen musst, bevor dein Code funktioniert.



5.1 Den Code entwerfen (Design & Edit)

Als erstes wirst du ein Programm schreiben, das die Nachricht „Hallo PH Graz!“ gefolgt von einem Bild auf dem Display deines micro:bit anzeigt. Danach soll „Hello world“ auf der REPL-Befehlszeile ausgegeben werden.

Es ist ratsam, darüber nachzudenken, was du mit deinem Code erreichen willst und wie du es erreichen willst, bevor du mit dem Schreiben beginnst. Es gibt hier zwar nicht viel zu planen oder entwerfen, aber nur, damit du verstehst, wie so ein Plan aussehen könnte:

```
Für immer wiederholen:
  Scrolle "Hallo PH Graz" über das LED Display
  Ein lachendes Gesicht anzeigen
  Gib "Hello world!" auf der Konsole aus
  Für zwei Sekunden warten
```

Es gibt zwei Möglichkeiten, die Ausgabe deines Codes anzuzeigen: Entweder du nutzt die auf dem micro:bit verfügbaren Ausgaben (z.B. die LEDs) oder die REPL-Konsole, die im Editor mit der `print` Anweisung ansteuerbar ist. Die Konsole ist besonders nützlich, um Bugs (Fehler) in deinem Code zu finden oder neue Konzepte oder Ideen auszuprobieren.

Gehen wir den Code, der unsere Planung in Python realisiert, Zeile für Zeile durch:

```
from microbit import *
```

Der Import von Paketen (wie microbit) in Python ermöglicht es uns, Funktionen oder Objekte zu verwenden, die in reinem Python nicht definiert sind. In unserem Fall sind es zum Beispiel die Funktionen `display` oder `show`.

```
while True:
```

In diesem Fall soll etwas (was auch immer dieser Anweisung folgt und eingerückt ist) ausgeführt werden, während die Bedingung, die auf `while` folgt, wahr ist. In diesem Fall ist die Bedingung das Schlüsselwort `True`, was bedeutet, dass diese Schleife ewig weiterläuft. Es ist dasselbe, als wenn du zB „Mach das solange wie 5 > 1 ist“ schreiben würdest, was ja für immer und ewig der Fall sein wird.

Der Rest des Programms ist ganz einfach

```
    display.scroll('Hallo PH Graz')
display.show(Image.HAPPY)
    print('Hello world!')
sleep(2000)
```

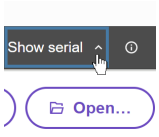
Es wird `Hallo PH Graz` und dann das lachende Gesicht auf dem LED Display angezeigt. Die Anweisung `print('Hello world!')`, gibt die Nachricht in der REPL-Konsole aus.

5.2 Programm auf den micro:bit laden (Flashen)

Gib den Code ein und klicke dann auf die Taste `Send to micro:bit`. Schau genau, ob alles so abläuft wie geplant.

Das Ergebnis auf dem micro:bit sollte in etwa so aussehen wie in der simulierten Vorschau.

Um das REPL-Fenster des micro:bit anzuzeigen, musst du die untere `Show Serial`-Schaltfläche anklicken:



Das REPL-Fenster zeigt uns Nachrichten vom micro:bit an und erlaubt uns auch, Befehle direkt an den micro:bit zu senden. Für den Moment werden wir die REPL nur benutzen um Nachrichten und Fehlermeldungen zu sehen, die wir mit dem `print`-Befehl ausgeben.

5.3 Etwas ändern

Der beste Weg, um zu lernen, wie etwas funktioniert, ist zu versuchen, den Code ein wenig zu verändern, um zu sehen was dann passiert. Und die *Dokumentation zu lesen*, um zu sehen, was alles möglich ist, gehört sowieso auch zum Alltag von Programmierern.

Fragen, die du dir stellen könntest:

- Wozu ist die Verzögerung (`sleep()`) da? Ist sie notwendig? Versuche einmal sie zu entfernen.
- Was passiert, wenn du `True` durch `False` ersetzst?
- Was passiert, wenn du `scroll()` durch `show()` ersetzst?

5.4 Überblick über die Arbeitsweise

Um Programme für den micro:bit schreiben zu können, musst du wissen, welche Befehle er verstehen kann und wie man diese in Python hinschreiben muss. Da man sich die ganzen Regeln und Begriffe (vor allem am Anfang) schwer merken kann, muss man wissen wo man nachschauen kann. Du wirst also sehr oft zwischen diesem Tutorial und dem **Web-Editor** hin- und herwechseln, oder in der Referenz nachsehen, wenn etwas unklar ist.

Da in diesem Tutorial nur das allernötigste zu Python beschrieben wird, um mit dem micro:bit arbeiten zu können, sind auch noch andere Unterlagen vorhanden, mit denen man noch mehr über Python erfahren kann.

Siehe auch:

- Schau dir auch die komplette micro:bit Dokumentation für MicroPython an.
- Um die Programmiersprache Python besser kennenzulernen stehen auch interaktive Jupyter-Notebooks zur Verfügung. Hier kannst du direkt Dinge ausprobieren und lernst dabei, wie Python 3 funktioniert.

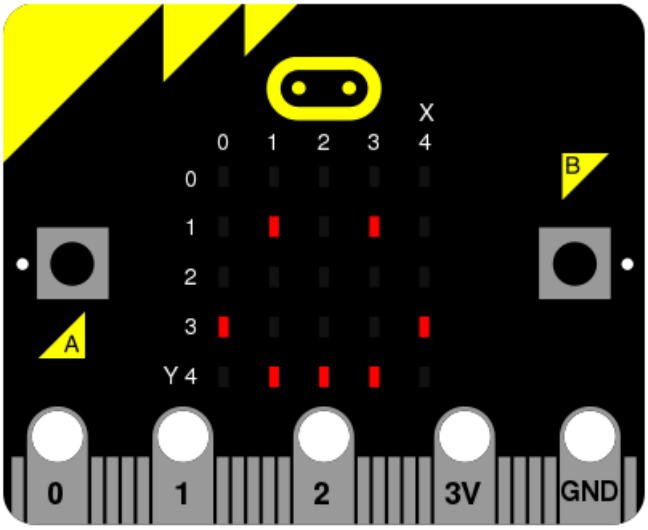
Nun hast du dein erstes Programm geschrieben und damit herumexperimentiert. In den nächsten Abschnitten erfährst du mehr über das Schreiben komplexerer Programme und über weitere Einsatzmöglichkeiten des micro:bit.

KAPITEL 6

LED Display

Dies ist eine Kurzanleitung für einige Dinge, die du mit dem LED Display machen kannst. Probiere die Dinge aus - schau, was passiert und was du alles machen kannst. Es gibt 25 LEDs, nummeriert von (0,0) in der oberen linken Ecke bis (4,4) in der unteren rechten Ecke und sie können alle auf verschiedene Helligkeitsstufen eingestellt werden.

Du kannst die LEDs wie einen Bildschirm benutzen, um einzelne Zeichen, eine Zeichenkette oder ein kleines Bild anzuzeigen.



6.1 Grundfunktionen

6.1.1 Einen String oder ein Bild anzeigen

Du kannst Zeichen - sogenannte `Strings` - oder Bilder auf dem LED Display mit der Methode `display.show()` anzeigen:

```
from microbit import *

display.show("Hallo")
```

Die Zeichen, die du auf dem Display anzeigst, müssen als `Strings` innerhalb eines Paares von Anführungszeichen stehen, entweder " " oder ' '.

Bilder benötigen keine Anführungszeichen, da sie ja keine `Strings` sind. Das `microbit` Modul hat schon viele Bilder eingebaut, die auf dem Display angezeigt werden können.

Um zum Beispiel eine lächelndes Gesicht anzuzeigen, kannst du das vorhandene `Image`-Objekt verwenden. Die vorhandenen Bilder werden dir von der Autovervollständigung im Mu-Editor zur Auswahl vorgeschlagen, nachdem du einen Punkt nach `Image.` eingegeben hast.

```
from microbit import *

display.show(Image.HAPPY)
```

Hier sind einige der anderen Bilder, die du verwenden kannst:

- `Image.HEART`, `Image.HEART_SMALL`
- `Image.HAPPY`, `Image.SMILE`, `Image.SAD`, `Image.CONFUSED`, `Image.ANGRY`, `Image.ASLEEP`, `Image.SURPRISED`, `Image.SILLY`, `Image.FABULOUS`, `Image.MEH`, `Image.YES`, `Image.NO`
- `Image.ARROW_N`, `Image.ARROW_NE`, `Image.ARROW_E`, `Image.ARROW_SE`, `Image.ARROW_S`, `Image.ARROW_SW`, `Image.ARROW_W`, `Image.ARROW_NW`
- `Image.MUSIC_CROTCHET`, `Image.MUSIC_QUAVER`, `Image.MUSIC_QUAVERS`
- `Image.XMAS`, `Image.PACMAN`, `Image.TARGET`, `Image.ROLLERSKATE`, `Image.STICKFIGURE`, `Image.GHOST`, `Image.SWORD`, `Image.UMBRELLA`
- `Image.RABBIT`, `Image.COW`, `Image.DUCK`, `Image.HOUSE`, `Image.TORTOISE`, `Image.BUTTERFLY`, `Image.GIRAFFE`, `Image.SNAKE`

6.1.2 Eine Laufschrift anzeigen

Benutze `scroll`, um einen String am Display als Laufschrift anzuzeigen

```
from microbit import *

display.scroll("Hallo!")
```

6.1.3 Eine Zahl anzeigen

Inzwischen zeigt der Befehl `display.show(6)` wirklich die Zahl 6 an, obwohl sie nicht in Anführungszeichen steht. Normalerweise führt das zu einem Fehler namens *TypeError*, da wir weiter oben ja gelernt haben, dass ein String and Bytes literals-Wert erwartet wird. Das ist zwar ganz praktisch, aber völlig untypisch für Python. Du solltest dich deshalb nicht darauf verlassen und **Zahlen vorher immer in Strings umwandeln**!

Das geht ganz einfach mit der `str()`-Methode. Probiere die unterschiedlichen `show`-Aufrufe aus, indem du die jeweilige Zeile „auskommentierst“ (dh die Raute # am Zeilenanfang löschst)

```
from microbit import *

display.show(str(6))       # korrekte Typumwandlung
display.show(6)            # funktioniert leider auch
#display.show(str(3.14))    # korrekte Typumwandlung
#display.show(3.14)         # funktioniert leider auch
#display.show("3,14")       # Vorsicht: Dezimalzeichen ist der Punkt!! Hier egal weil␣
↪String.
#display.show(31,1408)      # leider kein TypeError: Anzeige 31 mit 1408ms Verzögerung␣
↪zw. 3 und 1
#display.show(str(3,14))    # TypeError
```

Dass die Methode `show()` „anwenderfreundlicher“ gemacht wurde, führt im Zusammenhang mit Zahlen unter Umständen zu schwer nachvollziehbarem Verhalten - also Vorsicht!

6.1.4 Löschen des Displays

Wenn du das LED Display löschen möchtest, kannst du dies wie folgt tun:

```
from microbit import *

display.clear()
```

6.2 Fortgeschrittene Funktionen

6.2.1 Ein Pixel setzen

Du kannst die Helligkeit eines Pixels auf dem LED Display mit der Methode `set_pixel(spalte,zeile, hellligkeit)` einstellen oder ein- und ausschalten, indem du die Koordinaten eines Pixels *(x-Spalte,y-Zeile)* verwendest:

```
from microbit import *

display.set_pixel(0,4,9)
```

Das setzt die LED in Spalte `0` und Zeile `4` auf eine Helligkeit von `9`. Der Helligkeitswert kann eine ganze Zahl zwischen 0 und 9 sein. 0 schaltet die LED aus und 9 ist die hellste Einstellung.

Der folgende Code setzt mit einer `For Schleife` jedes der Pixel in der oberen Zeile (y=0) auf volle Helligkeit (9):

```
from microbit import *

for x in range(5):
    display.set_pixel(x,0,9)
    sleep(500)
```

Aufgaben:

- Passe den Code aus dem Beispiel so an, dass er die Pixel der ersten Spalte statt der ersten Zeile zum Leuchten bringt.
- Passe das Programm so an, dass die mittlere Pixelreihe anstelle der oberen Reihe eingeschaltet wird.
- Schreibe ein Programm, das das mittlere Pixel jede halbe Sekunde an- und ausblinken lässt.

Du könntest verschachtelete *For-Schleifen* verwenden, um alle LEDs nacheinander einzustellen:

```
from microbit import *

display.clear()
for y in range(0, 5):
    for x in range(0, 5):
        display.set_pixel(x,y,9)
        sleep(100)
```

Die `for`- Schleife lässt dich eine Schleife mit Hilfe eines Zählers eine bestimmte Anzahl von Malen ausführen. Die **äußere Schleife**:

```
for y in range(0,5)
```

führt die Schleife fünfmal aus und ersetzt `y` durch aufeinanderfolgende Werte im Bereich `0` bis `4` für `y`. Die Schleife hört auf, bevor sie den letzten Wert im angegebenen Bereich erreicht.

Die **innere Schleife**:

```
for x in range(0,5):
```

führt die Schleife fünfmal aus und ersetzt `x` jedes Mal durch aufeinanderfolgende Werte aus dem Bereich `0` bis `4`. So werden der Reihe nach also für jede Zeile alle Pixel gesetzt.

Aufgabe:

- Wie muss die For Schleife aussehen, damit die Pixel spaltenweise gesetzt werden?
- Erweitere den Beispielcode, damit die Pixel auch wieder ausgeschaltet werden. Um sie in umgekehrter Reihenfolge auszuschalten, brauchst du eine Schleife, die rückwärts zählt. Ein Beispiel dafür wäre, `for x in range(4,-1,-1):`
- Schreibe ein Programm, das die einzelnen Pixel des Displays ein- und ausblinken lässt. Es sollte so aussehen, als würde das blinkende Pixel über die ganze Matrix wandern.
- Schreibe ein möglichst kurzes Programm, um am Display ein Quadrat kleiner und größer werden zu lassen.

6.2.2 Eigene Bilder

Was ist, wenn du dein eigenes Bild für das Display auf dem micro:bit erstellen möchtest?

Wie bereits erwähnt, kann jedes LED-Pixel auf dem Display auf einen von zehn Werten von 0 (aus) bis 9 (voll ein) eingestellt werden. Mit dieser Information ist es möglich, ein neues Bild wie dieses zu erstellen

```
from microbit import *

boot = Image("05050:"
             "05050:"
             "05050:"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
                "99999:"
                "09990")

display.show(boot)
```

Eigentlich brauchst du das nicht über mehrere Zeilen zu schreiben. Wenn du den Überblick über jede der Zeilen behältst, kannst du den Code so umschreiben:

```
boot = Image("05050:05050:05050:99999:09990")
```

(Wenn es funktioniert, sollte das Gerät ein altmodisches „Blue Peter“ Segelschiff anzeigen wobei die Masten dunkler sind als der Rumpf des Schiffes).

Weißt du jetzt, wie man ein Bild malt? Was du dir merken solltest ist, dass jede Zeile des Displays durch eine Zeile aus Zahlen dargestellt wird, die mit `:` endet und zwischen `"` Anführungszeichen eingeschlossen ist!

Jede Zahl gibt eine Helligkeit an. Es gibt fünf Zeilen mit fünf Zahlen, so dass es möglich ist, die individuelle Helligkeit für jedes der fünf Pixel auf jeder der fünf Zeilen auf dem Display einzustellen und anzuzeigen.

Welches Bild ist hier zu sehen?

```
from microbit import *

meinBild = Image("00900:"
             "09090:"
             "90009:"
             "05550:"
             "05950")

display.show(meinBild)
```

6.2.3 Animation

Um eine Animation zu machen, verwendet man am einfachsten eine Liste von Bildern.

Wir können das anhand von bereits eingebauten Listen demonstrieren - `Image.ALL_CLOCKS` und `Image.ALL_ARROWS`:

```
from microbit import *

display.show(Image.ALL_CLOCKS, loop=True, delay=100)
```

Der micro:bit zeigt jedes Bild in der Liste an, eines nach dem anderen. Wenn du `loop=True` einstellst, wird das Programm in einer Schleife durch die Liste laufen, ohne Ende. Es ist auch möglich eine Verzögerung zwischen den Bildern einzustellen, indem man das Attribut `delay` auf den gewünschten Wert in Millisekunden setzt `delay=100`.

Um deine eigene Animation zu erstellen, musst du also nur eine Liste von Bildern erstellen.

In diesem Beispiel wird ein Boot im Boden des Displays versinken. Dazu haben wir eine Liste mit 6 Bootsbildern definiert:

```
from microbit import *

boat1 = Image("05050:"
              "05050:"
              "05050:"
              "99999:"
              "09990")
```

(Fortsetzung auf der nächsten Seite)

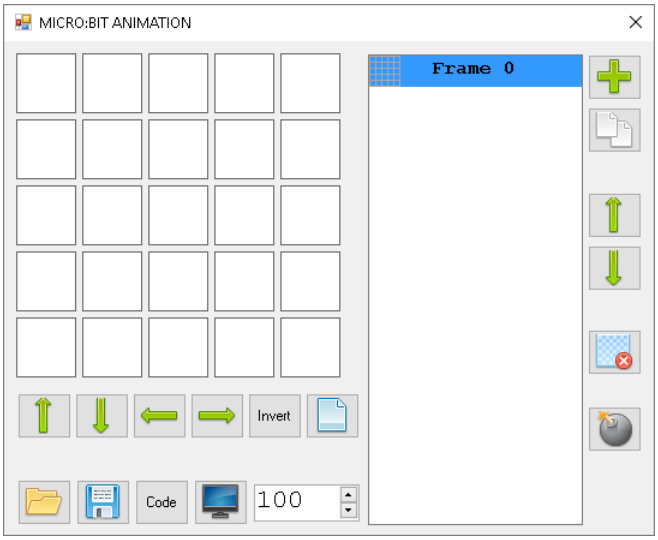
(Fortsetzung der vorherigen Seite)

```
boat2 = Image("00000:"
              "05050:"
              "05050:"
              "05050:"
              "99999")

boat3 = Image("00000:"
              "00000:"
              "05050:"
              "05050:"
              "05050")

boat4 = Image("00000:"
              "00000:"
              "00000:"
              "05050:"
              "05050")

boat5 = Image("00000:"
              "00000:"
              "00000:"
              "00000:"
              "05050")

boat6 = Image("00000:"
              "00000:"
              "00000:"
              "00000:"
              "00000")

all_boats = [boat1, boat2, boat3, boat4, boat5, boat6] #Liste aller Boote
display.show(all_boats, delay=200)
```

6.3 Übungsaufgaben

- Probiere einige der eingebauten Bilder aus, um zu sehen, wie sie aussehen.
- Animiere die `Image.ALL_ARROWS` Liste. Wie vermeidest du eine ewige Schleife (Hinweis: das Gegenteil von `True` ist `False`). Kannst du die Geschwindigkeit der Animation verändern?
- Erstelle dein eigenes Bild. Versuche als nächstes, es aus- und wieder einzublenden?
- Programmiere einen Würfel, der zufällig eines der 6 Würfelmuster anzeigt
- Mache ein Sprite, benutze eine einzelne LED auf dem Display. Kannst du es springen lassen, wenn du eine Taste drückst?

Tipp: Auf der sehr empfehlenswerten Seite MultiWingSpan kann man ein kleines Tool herunterladen, mit dem man den Code für solche micro:bit Bilder und Animationen ganz einfach erstellen kann!


MICRO:BIT ANIMATION
Frame 0
Invert
Code
100

KAPITEL 7

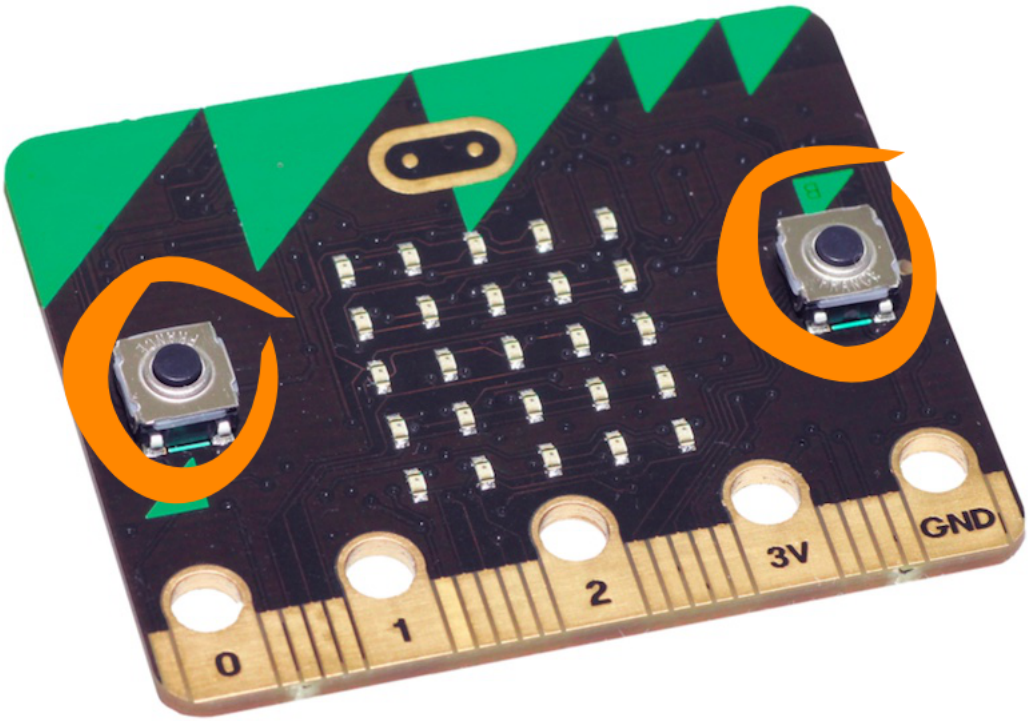
Tasten

Bis jetzt haben wir Code erstellt, der das Gerät dazu bringt, einfach irgendetwas zu machen und auszugeben. Wir reden davon, dass ein *Output* bzw. eine *Ausgabe* erzeugt wurde. Wir wollen aber auch, dass das Gerät auf Ereignisse von Außen reagiert. Solche Ereignisse werden *Inputs* oder *Eingaben* genannt.

Es ist leicht zu merken: Output ist das, was das Gerät in die Welt hinausgibt, während Input das ist, was in das Gerät hineingeht, um dann verarbeiten zu werden.

Man spricht auch vom **EVA**-Prinzip: **E** ingabe - **V** erarbeitung - **A** usgabe

Das augenscheinlichste Eingabegerät auf dem micro:bit sind die beiden Tasten, die mit `A` und `B` bezeichnet sind. Du kannst die Tasten verwenden, um Eingaben vom Benutzer zu erhalten.



Irgendwie sollte MicroPython auf das Drücken der Tasten reagieren können. Vielleicht möchtest du dein Programm

mit einem Tastendruck starten oder stoppen oder vielleicht möchtest du wissen, wie oft jede Taste gedrückt wurde. All das ist recht einfach umzusetzen.

Link zur englischsprachigen MicroPython-Dokumentation

7.1 Grundfunktionen

Wenn wir wollen, dass MicroPython auf Tastendruck-Ereignisse reagiert, sollten wir das in eine Endlosschleife setzen und prüfen, ob die Taste zB. gedrückt (`is_pressed`) ist.

Eine Endlosschleife ist schnell mit einer While Schleife erstellt:

```
while True:
    # Mach etwas
```

(Erinnere dich, `while` prüft, ob etwas wahr (`True`) ist, um herauszufinden, ob es seinen Codeblock ausführen soll. Da `True` offensichtlich immer *Wahr* ist, erhältst du eine unendliche Schleife).

7.1.1 Prüfen, ob eine Taste gedrückt ist

Manchmal wollen wir nur, dass ein Programm wartet, bis etwas passiert. Zum Beispiel könnten wir den micro:bit anweisen, zu warten, bis, sagen wir, die Taste `A` gedrückt wird um dann eine Nachricht auszugeben. Das geht zum Beispiel so:

```
from microbit import *

while True:
        if button_a.is_pressed():
                display.scroll("A")
        else:
                display.show(Image.ASLEEP)
```

Das heißt, wenn die Taste `A` gedrückt wird, wird ein `A` und ansonsten `Image.ASLEEP` auf dem LED Display angezeigt.

Das Problem bei der Verwendung von `is_pressed()` ist, dass du, wenn du die Taste nicht genau im Moment der Abfrage drückst, nicht feststellen kannst, ob die Taste jemals gedrückt wurde oder nicht. Es könnte der Fall sein, dass der Benutzer die Taste drückt, während der Code etwas anderes macht, und der Tastendruck wird übersehen.

Die `was_pressed()` Funktion ist nützlich, wenn du Code schreiben willst, der gelegentlich prüft, ob die Taste gedrückt wurde, dazwischen aber etwas anderes macht. Auf diese Weise solltest du nie wieder einen Tastendruck verpassen:

```
from microbit import *

while True:
    if button_a.was_pressed():
        display.scroll("A")
    else:
        display.show(Image.ASLEEP)

    sleep(1000)
```

Am Display wird ein `A` für eine Sekunde angezeigt, wenn du die Taste drückst, und dann wird `Image.ASLEEP` angezeigt. Wenn du die Taste drückst, während das Programm gerade die Zeitverzögerung auslöst, dann wird das `A` nicht sofort angezeigt, sondern erst, wenn die Testbedingung der if-Anweisung ausgeführt wird. Das wird umso deutlicher, je länger du die Verzögerung einstellst.

Versuche nun `button_a.isPressed()` anstelle von `button_a.was_pressed()` zu verwenden.

7.1.2 Tamagotchi

Wir wollen als nächstes ein ganz einfaches Tamagotchi machen. Es ist immer traurig, außer du drückst die Taste `A`. Wenn du die Taste `B` drückst, stirbt es. (Mir ist klar, dass das kein sehr ausgefeiltes Spiel ist. Vielleicht fällt dir etwas ein, wie man es verbessern und netter gestalten kann):

```
from microbit import *

while True:
        if button_a.is_pressed():
                display.show(Image.HAPPY)
        elif button_b.is_pressed():
                break
        else:
                display.show(Image.SAD)

display.clear()
```

Um zu prüfen, welche Tasten gedrückt werden, benutzen wir `if` („wenn“), `elif` (kurz für „else if“ bzw. „sonst wenn“) und `else` („sonst“). Diese sogenannten *Bedingungen* sind ein wichtiger Bestandteil aller Programmiersprachen und funktionieren folgendermaßen:

```
Wenn etwas ist Wahr:
        # mach dies
SonstWenn etwas anderes ist Wahr:
        # mach das
Sonst:
        # mach sonstwas.
```

In Python ähnelt das sehr der gesprochenen englischen Sprache:

```
if something is True:
        # do one thing
elif some other thing is True:
        # do another thing
else:
        # do yet another thing.
```

Die Methode `is_pressed` liefert nur zwei Ergebnisse: `True` oder `False`. Wenn du die Taste drückst, gibt sie `True`, ansonsten gibt sie `False` zurück. Den obigen Code könnte man so ins Deutsche übersetzen:

„Für immer und ewig, wenn Taste A gedrückt wird, zeige ein glückliches Gesicht, oder, wenn Taste B gedrückt wird, beende die Schleife und damit das Spiel. Immer sonst zeige ein trauriges Gesicht.“

Mit der `break` Anweisung „brechen“ wir aus der Schleife aus und stoppen das eigentlich für immer und ewig laufende Programm.

Ganz am Ende, wenn das Tamagotchi tot ist, löschen (`clear`) wir das Display.

- Fällt dir ein Weg ein, dieses Spiel weniger tragisch zu gestalten?
- Wie würdest du überprüfen, ob *beide* Tasten gedrückt sind? (Tipp: weiter unten wird das behandelt).

7.1.3 Zählen der Anzahl der Tastendrücke

Um zu zählen, wie oft eine Taste gedrückt wurde, kannst du die `get_presses()` Methode verwenden. Hier ist ein Beispiel:

```
from microbit import *

while True:
   sleep(3000)
   count = button_a.get_presses()
   display.scroll(str(count))
```

Der micro:bit pausiert für 3 Sekunden, wacht dann auf und überprüft, wie oft die Taste `A` gedrückt wurde. Die Anzahl der Tastendrücke wird in `count` gespeichert.

Um `count` am Display auszugeben, muss man beachten, dass es sich dabei um eine Zahl - die Anzahl der Tastendrücke - handelt. `scroll` kann aber nur Strings ausgeben, weshalb wir den numerischen Wert `count` in einen String aus Zeichen umwandeln müssen. Das machen wir mit der `str` Funktion (kurz für „string“ ~ sie wandelt alle möglichen Objekte in Strings um).

Kannst du deine eigene `get_presses` Funktion erstellen?

7.2 Erweiterte Funktionen

7.2.1 Überprüfung beider Tasten

Es ist möglich, eine Reihe von Ereignissen mit Hilfe von bedingten Anweisungen zu überprüfen. Sagen wir, du möchtest prüfen, ob die Taste `A` gedrückt wurde oder die Taste `B` gedrückt wurde oder ob beide Tasten zur gleichen Zeit gedrückt wurden:

```
from microbit import *

while True:
    if button_a.is_pressed() and button_b.is_pressed():
        display.scroll("AB")
        break
    elif button_a.is_pressed():
        display.scroll("A")
    elif button_b.is_pressed():
        display.scroll("B")
    sleep(100)
```

Der obige Code zeigt den Buchstaben an, der der Taste entspricht. Wenn beide Tasten gleichzeitig gedrückt werden, wird `AB` angezeigt.

Was passiert, wenn `sleep(0)` gesetzt bzw. ganz weggelassen wird?

7.3 Übungsaufgaben

- Ändere, was angezeigt wird, wenn du eine Taste drückst.
- Spiele, die Benutzereingaben benötigen.

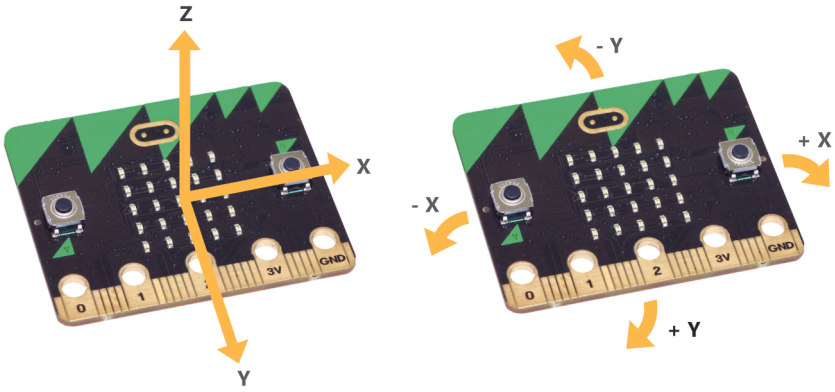
KAPITEL 8

Beschleunigungsmesser

Wie der Name schon sagt, misst der Beschleunigungsmesser die Beschleunigung oder anders gesagt, die Bewegungsänderungen deines BBC micro:bit. Der Beschleunigungsmesser ist so eingestellt, dass er Beschleunigungswerte im Bereich von +2g bis -2g misst (mit g wird die Größe der Erdbeschleunigung bzw. Gravitation bezeichnet), und diese Werte können mit MicroPython ausgelesen und auf den Bereich 0 … 1024 abgebildet werden.

Der micro:bit misst die Bewegung entlang dreier Achsen:

- X - Bewegung/Kippen von links nach rechts.
- Y - Bewegung/Kippen nach vorne und hinten.
- Z - Bewegung nach oben und unten.



8.1 Grundfunktionen

Die Messung für jede Achse ist eine positive oder negative Zahl die einen Wert in milli-g's angibt. Wenn der Messwert 0 ist, bist du „ruhig" entlang der jeweiligen Achse.

Du kannst die Beschleunigungswerte einzeln oder alle drei Werte auf einmal abrufen und sie in einer **Liste** speichern. Du kannst mehr über Listen in den Grundlagen der Programmierung lernen, aber für den Moment verwende einfach den folgenden Code:

```
from microbit import *

while True:
    x = accelerometer.get_x()
    y = accelerometer.get_y()
    z = accelerometer.get_z()
    print("x, y, z:", x, y, z)
    sleep(500)
```

Lade den Code auf den micro:bit und öffne den seriellen Monitor. Halte den micro:bit flach mit den LEDs nach oben. Du solltest sehen, dass die X- und Y-Beschleunigungen nahe Null sind, und die Z-Beschleunigung nahe bei -1024. Das bedeutet, dass die Schwerkraft relativ zum micro:bit nach unten wirkt. Drehe das Board um so dass die LEDs dem Boden am nächsten sind. Der Z-Wert sollte positiv werden bei +1024 milli-g. Wenn du den micro:bit kräftig genug schüttelst, wirst du sehen, dass die Beschleunigungen bis zu $\pm$2048 milli-g ansteigen. Das liegt daran, dass der Beschleunigungssensor auf ein Maximum von $\pm$2048 milli-g eingestellt ist: die tatsächliche Zahl kann höher sein als das.

Wenn du dich jemals gefragt hast, woher ein Mobiltelefon weiß, in welche Richtung es den Bildschirm ausrichten soll, dann liegt das daran, dass es den Beschleunigungssensor auf genau die gleiche Weise wie das obige Programm abfragt. Auch Gamecontroller enthalten Beschleunigungssensoren, um die Steuerung zu ermöglichen.

8.1.1 Gesten

Eine wirklich interessante Anwendung des Beschleunigungssensors ist die Gestenerkennung. Wenn du deinen BBC micro:bit auf eine bestimmte Art und Weise bewegst (als Geste), dann ist der micro:bit in der Lage, dies zu erkennen.

Der micro:bit ist in der Lage, die folgenden Gesten zu erkennen:

- `up`, `down`
- `left`, `right`
- `face up`, `face down`
- `freefall`, `3g`, `6g`, `8g`

- `shake`.

Gesten werden immer als Strings dargestellt. Während die meisten Namen offensichtlich sein sollten, gelten die `3g`, `6g` und `8g` Gesten, wenn das Gerät entsprechend schnell beschleunigt wird.

Um die aktuelle Geste zu erhalten, benutze die Methode `accelerometer.current_gesture()`. Das Ergebnis wird eine der oben genannten Gesten sein.

Zum Beispiel wird dieses Programm ein glückliches Emoticon anzeigen, wenn das Display nach oben zeigt:

```
from microbit import *

while True:
    geste = accelerometer.current_gesture()
    if geste == "face up":
        display.show(Image.HAPPY)
    else:
        display.show(Image.ANGRY)
```

1. Innerhalb des *Geltungsbereichs (Scope)* der Schleife wird die aktuelle Geste gelesen und der Variablen `geste` zugewiesen.
2. Die `if`-Bedingung prüft, ob `geste` gleich `"face up"` ist (Python verwendet ==, um auf Gleichheit zu testen, ein einzelnes Gleichheitszeichen = wird für die Zuweisung verwendet - so wie wir die gelesenen Gesten in der Zeile darüber der Variablen `geste` zugewiesen haben).
3. Wenn die Geste gleich `"face up"` ist, dann benutze das Display, um ein glückliches Gesicht zu zeigen.
4. Ansonsten wird das Gerät dazu gebracht, wütend dreinzuschauen!

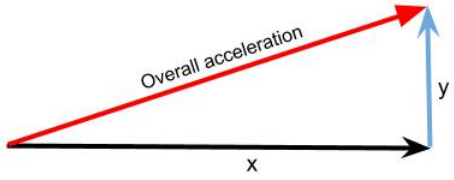
Was macht das folgende Programm?:

```
from microbit import *

while True:
        if accelerometer.is_gesture("up"):
                display.show(Image.ARROW_S)
        elif accelerometer.is_gesture("right"):
                display.show(Image.ARROW_E)
        elif accelerometer.is_gesture("down"):
                display.show(Image.ARROW_N)
        elif accelerometer.is_gesture("left"):
                display.show(Image.ARROW_W)
        else:
                display.clear()
        sleep(20)
```

8.2 Fortgeschrittene Funktionen

Für den Beschleunigungssensor gibt es keine, aber es lohnt sich zu schauen, wie wir die 3D-Beschleunigung nutzen können, um verschiedene Arten von Bewegung zu erkennen.

Wir könnten zB erkennen wollen, ob unser micro:bit geschüttelt wird. Die Beschleunigung ist eine sogenannte Vektorgröße - sie hat einen Betrag (Größe, Länge) und eine Richtung. Um den Gesamtbetrag in X- und Y-Richtung zu erhalten, ohne auf die Z-Achse zu achten, könnte man bei dieser 2D-Beschleunigung dann wie folgt vorgehen:



Wir können den Betrag (Länge) der Resultierenden mit dem Satz des Pythagoras berechnen:

$$beschleunigung = \sqrt{x^2 + y^2}$$

Das gleiche Prinzip gilt für einen 3D-Beschleunigungsmesser. Der Gesamtbetrag des resultierenden Beschleunigungsvektors ist also gleich:

$$beschleunigung = \sqrt{x^2 + y^2 + z^2}$$

Berechnung der Gesamtbeschleunigung:

```
from microbit import *
import math

while True:
    x = accelerometer.get_x()
    y = accelerometer.get_y()
    z = accelerometer.get_z()
    beschleunigung = math.sqrt(x**2 + y**2 + z**2)
    print("Beschleunigung", beschleunigung)
    sleep(500)
```

Wenn du den Beschleunigungssensor still hältst (auf den Tisch legst), ergibt dies eine Beschleunigung von etwa 1g, unabhängig davon, in welcher Orientierung du den BBC micro:bit hältst. Wenn du nun den micro:bit bewegst, wird sich dieser Wert ändern und davon abweichen. Tatsächlich wird der Wert leicht variieren, auch wenn du ihn still hältst, weil der Beschleunigungsmesser kein perfektes Messgerät ist.

Immer wenn wir eine Größe genau wissen wollen, ist eine sogenannte *Kalibrierung* nötig, bei der die Sensordaten genau eingemessen und mit einem Richtwert verglichen werden.

8.3 Übungsaufgaben

- Benutze die BBC micro:bit Musikbibliothek und spiele eine Note, die auf dem Messwert des Beschleunigungsmessers basiert. Tipp: Stelle die Tonhöhe auf den Wert des Beschleunigungsmessers ein.
- Zeige die Zeichen ‚L' oder ‚R' an, je nachdem, ob der BBC micro:bit nach links oder rechts gekippt ist.
- Lasse die LEDs aufleuchten, wenn die Größe der Beschleunigung größer als 1024 milli-g's ist.
- Schüttle den micro:bit, um die LEDs aufleuchten zu lassen.
- Mache einen Würfel. Tipp: benutze eine der Python Zufallsfunktionen. Gib `import random` am Anfang deines Programms ein und verwende `random.randrange(start, stop)`. Dies wird eine Zufallszahl zwischen `start` und `stop - 1` erzeugen.

KAPITEL 9

Kompass

Ein Magnetometer misst die magnetische Feldstärke in jeder der drei Achsen. Es kann verwendet werden, um einen digitalen Kompass zu erstellen oder um Magnetfelder zu erforschen, wie die, die von einem Permanentmagneten erzeugt werden oder die um eine Spule, durch die ein Strom fließt, entstehen.

Die Auswertung der magnetischen Feldstärke ist nicht einfach. Der Treiber für das Magnetometer gibt Rohwerte zurück. Jedes Magnetometer ist anders und muss kalibriert werden, um Abweichungen in den Rohwerten und Verzerrungen durch das Magnetfeld zu berücksichtigen, die durch die sogenannte Hart- und Weicheiseninterferenz entstehen.

Bevor du zu messen beginnst, solltest du deinen BBC micro:bit also unbedingt jedesmal kalibrieren, aber beachte dabei folgendes:

Warnung: Das Kalibrieren des Kompasses führt dazu, dass dein Programm pausiert, bis die Kalibrierung abgeschlossen ist. Die Kalibrierung besteht aus einem kleinen Spiel, um durch Drehen des Geräts einen Kreis auf das LED Display zu zeichnen.

9.1 Grundfunktionen

Die *Schnittstelle* des Magnetometers (die Art und Weise, wie Daten ausgetauscht werden) sieht sehr ähnlich aus wie die des Beschleunigungsmessers, außer dass wir nur die x- und y-Werte zur Richtungsbestimmung verwenden.

Denke daran, bevor du den Kompass benutzt, solltest du ihn kalibrieren, sonst können die Messwerte falsch sein:

```
from microbit import *

compass.calibrate()

while True:
    x = compass.get_x()
    y = compass.get_y()
    print("x Wert: ", x, ", y Wert: ", y)
    sleep(500)
```

Dieser Code liest das Magnetfeld in zwei Richtungen (wie ein echter Kompass) aus und gibt dann die Werte zurück, was einfacher aussieht als es ist. Je stärker das Feld ist, desto größer ist die Zahl. Experimentiere damit und finde heraus, welches die x-Achse für das Magnetometer ist.

Die oberen Werte sind nicht sehr sinnvoll einsetzbar. Der micro:bit kann nach der Kalibrierung aber zum Glück die genaue Richtung zurückgeben, in die er gedreht ist:

```
compass.heading()
```

Das gibt die Kompassrichtung als Ganzzahl im Bereich von 0 bis 360 an, was dem Winkel in Grad im Uhrzeigersinn entspricht. Norden wäre also 0. Du musst das Gerät erst kalibrieren, bevor du `compass.heading()` verwendest.:

```
from microbit import *

compass.calibrate()
while True:
    richtung = compass.heading()
    print(richtung)
    sleep(100)
```

Mit diesem Programm kannst du testen, welche Werte wir für die Kompassrichtung erhalten. Achte darauf, dass du auf die REPL Taste klickst, nachdem du dieses Programm auf den micro:bit flashst. Wenn du die Kalibrierung abgeschlossen hast, solltest du 10 Positionsangaben pro Sekunde auf der Konsole ausgegeben bekommen.

9.2 Übungsaufgaben

- Mache den micro:bit zu einem Kompass, der die LED aufleuchten lässt, die dem Norden am nächsten liegt.
- Kalibriere dein Magnetometer. Finde heraus, ob die Kalibrierung über die Zeit (ungefähr) gleich bleibt und ob sie innerhalb oder außerhalb eines Gebäudes oder in der Nähe von etwas, das viel Stahl enthält (z.B. ein Aufzug), gleich ist.

KAPITEL 10

Thermometer

Das Thermometer auf dem micro:bit ist in einem der Chips eingebettet - und Chips werden warm, wenn sie eingeschaltet werden. Deshalb misst es die Raumtemperatur nicht sehr genau. Der Chip, mit dem die Temperatur gemessen wird, befindet sich auf der Rückseite des micro:bit links:

10.1 Grundfunktionen

Das Thermometer hat nur eine grundlegende Funktion - die Temperatur zu messen, die als Ganzzahl in Grad Celsius zurückgegeben wird:

```
from microbit import *

while True:
    temp = temperature()
    display.scroll(str(temp) + 'C')
    sleep(500)
```

Die Temperatur, die das Thermometer misst, wird typischerweise höher sein als die tatsächliche Temperatur, da es sowohl vom Raum als auch von der Elektronik auf dem Board erwärmt wird. Wenn wir wissen, dass die Temperatur 27°C beträgt, aber das micro:bit ständig Temperaturen anzeigt, die, sagen wir, 3 Grad höher sind, dann können wir die Messung korrigieren. Dazu musst du die wirkliche Temperatur kennen bzw. ohne den micro:bit mit einem anderen Thermometer messen.

Wie könnte das aussehen?

10.2 Übungsaufgaben

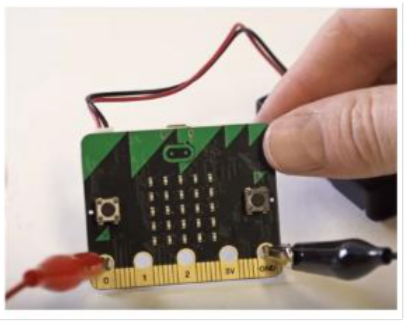
- Versuche das Thermometer zu kalibrieren. Zeigt es immer noch die richtige Temperatur an, wenn du es an einen wärmeren oder kühleren Ort stellst?
- Lass die LEDs ihr Muster ändern, wenn sich die Temperatur ändert.
- Finde heraus, wie sehr sich die Temperatur in einem Raum ändert, wenn du ein Fenster öffnest - was denkst du, sagt das über die verschwendete Heizenergie aus?

KAPITEL 11

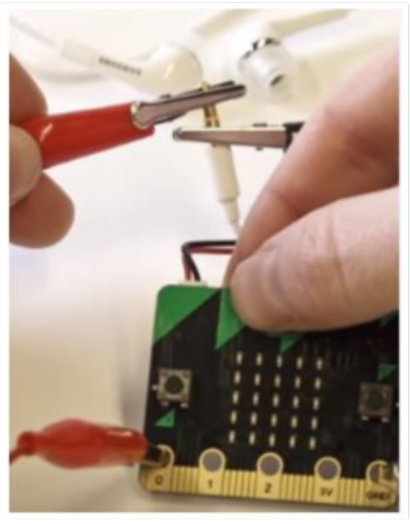
Musik

Der **micro:bit V2** kann direkt zum Abspielen einfacher Melodien verwendet werden. Auch das ältere Modell ist dazu in der Lage, vorausgesetzt du verbindest einen Lautsprecher mit deinem Board.

Wenn du Kopfhörer verwendest, kannst du Krokodilklemmen verwenden, um deinen micro:bit mit Kopfhörern zu verbinden:



Connect crocodile clips to pin 0 and GND respectively



Connect the clip leading from the ground pin to the base and clip from pin 0 to the tip of the headphone jack

Warnung: Du kannst die Lautstärke des Tonpegels nicht über den micro:bit steuern. Bitte sei sehr vorsichtig, wenn du Kopfhörer verwendest. Ein Lautsprecher ist die sicherere Wahl bei der Arbeit mit Klängen.

Du kannst deinen micro:bit auch mittels Krokodilklemmen mit einem Lautsprecher verbinden:

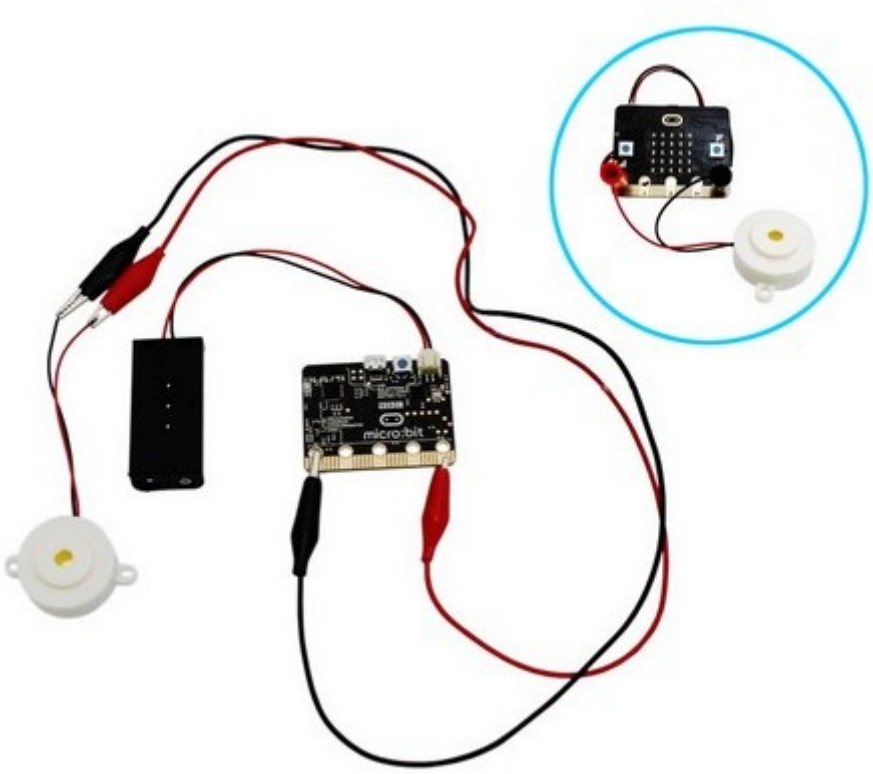


Abb. 1: Source: <https://www.kitronik.co.uk/blog/microbit-alarm-kitronik-university/>

11.1 Grundfunktionen

11.1.1 Eine Melodie spielen

Um eine Melodie zu spielen, kannst du die Funktion `play` verwenden:

```
from microbit import *
import music

music.play(music.NYAN)
```

Bemerkung: Du musst das Modul `music` importieren, um Klänge zu spielen und zu regeln.

Das Musikmodul enthält eine Reihe von eingebauten Melodien. Hier sind einige von ihnen:

- `music.DADADADUM`
- `music.ENTERTAINER`
- `music.PRELUDE`
- `music.ODE`
- `music.NYAN`
- `music.RINGTONE`

11.1.2 Mach deine eigene Melodie

Um eine Melodie zu spielen, gibst du die zu spielende Note an (C,D,E,F,G,A,B; inkl. der Halbtöne (z.B.: C#)). Wahlweise kannst du auch die Oktave (1-8) und die Dauer der Wiedergabe angeben:

```
from microbit import *
import music

# Play a 'C'
music.play('C')

# Spiele ein 'C' 4 Schläge lang
music.play('C:4')

# Spiele 4 Schläge lang ein 'C' in der 3. Oktave
music.play('C3:4')
```

Eine Reihe von Noten nacheinander zu spielen ist einfach, du fügst die Noten, die du spielen willst, einfach in eine Liste ein:

```
from microbit import *
import music

# Lied: Frere Jacques
lied = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4", "C4:4",
        "E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]
music.play(lied)
```

Der Micro:bit merkt sich die Oktave der zuvor definierten Note. Daher kann die obige Melodie wie folgt umgeschrieben werden:

```
lied = ["C4:4", "D4:4", "E4:4", "C:4", "C:4", "D:4", "E:4", "C:4",
        "E:4", "F4:4", "G4:8", "E:4", "F:4", "G:8"]
```

11.2 Fortgeschrittene Funktionen

Du kannst auch die Note, die du spielen willst, über ihre Frequenz mit der Methode `pitch` bestimmen. Zum Beispiel, um einen Polizeisirenen-Effekt zu erzeugen

```
while True:
        for freq in range(880, 1760, 16):
                music.pitch(freq, 6)
        for freq in range(1760, 880, -16):
                music.pitch(freq, 6)
```

Kannst du erraten, was hier passiert? Jedes Mal, wenn du die Schleife durchläufst, wird eine neue Frequenz berechnet, indem du 16 addierst (oder subtrahierst). Eine for-Schleife die man hören kann!

11.3 Übungsaufgaben

- Erfinde deine eigene Melodie.
- Baue ein Musikinstrument. Verändere die Tonhöhe des gespielten Tons basierend auf den Messwerten des Beschleunigungsmessers.

11.4 Sprachausgabe

Weil es gar soviel Spaß macht, damit herumzuspielen, möchte ich hier auf das noch in Entwicklung befindliche `speech`-Modul hinweisen. Auf der Seite der englischsprachigen MicroPython-Dokumentation wird es vorgestellt, kann sich aber noch verändern.

Hier ein Versuch, unserem Engländer microbit ein paar deutsche Sätze zu entlocken:

```
from microbit import *
import speech

# Der Klassiker auf Deutsch in spezieller Lautschrift.
speech.pronounce("/HAHLLOH WEHLT.")

sleep(700)

# So geht's auch ;-)
speech.say("Eech been eyen clyner robotair",  speed=92, pitch=60, throat=190,␣
↪mouth=190)
```

KAPITEL 12

Funk

Der Micro:bit verfügt über eine Bluetooth Lowe Energy (BLE) Antenne, die zum Senden und Empfangen von Nachrichten genutzt werden kann.

Link zur englischsprachigen MicroPython-Dokumentation

12.1 Grundfunktionen

12.1.1 Vorbereitung

Bevor du den Funk benutzen kannst, darfst du nicht vergessen, das Modul `radio` zu importieren und den Funk mit `radio.on()` einzuschalten. Sobald der Funk eingeschaltet ist, kannst du damit Nachrichten von jedem anderen micro:bit in Reichweite empfangen:

```
from microbit import *
import radio

radio.on()
```

Da dieses Modul Strom verbraucht, sollte es, wenn es nicht mehr benötigt wird mittels `radio.off()` wieder ausschalten.

Einstellen des Kanals (Funkgruppe)

Wenn du nur Nachrichten innerhalb einer Gruppe von Geräten teilen willst, dann muss jeder micro:bit in der Gruppe so konfiguriert werden, dass er die gleiche Kanalnummer besitzt. Die Kanalnummer muss eine Zahl zwischen `0` und `100` sein:

```
# Wähle die Kanalnummer 19
radio.config(channel=19)
```

Es ist wichtig, dies zu tun, besonders wenn du dich in einem Raum mit anderen Leuten, die auch ihre micro:bits benutzen, befindest, weil dein micro:bit sonst alle Nachrichten in der Nähe mitbekommt und das ist nicht das, was du normalerweise willst.

Einstellen der Sendeleistung

Zum Schluss solltest du die Sendeleistung für den Funk einstellen. Standardmäßig sendet dein micro:bit auf der Leistungsstufe 0, was bedeutet, dass deine Nachrichten nicht sehr weit übertragen werden. Die Sendeleistung kann ein Wert zwischen `0` und `7` sein.:

```
# Sendeleistung auf 7 stellen
radio.config(power=7)
```

12.1.2 Eine Nachricht senden und empfangen

Jetzt bist du bereit, eine Nachricht zu senden oder zu empfangen. Du kannst eine Zeichenkette mit einer Länge von bis zu 250 Zeichen senden:

```
message_to_master = "Zwischen uns funkt es!"

radio.send(message_to_master)
```

Empfange eine Nachricht:

```
message_received = radio.receive()
```

12.1.3 Alle Teile zusammengefügt

```
from microbit import *
import radio

radio.on()
radio.config(channel=19)        # Wähle deine eigene Kanalnummer
radio.config(power=7)           # Setze das Signal auf die volle Stärke

message_to_master = "Zwischen uns funkt es!"

# Event loop.
while True:
        radio.send(message_to_master)
        incoming = radio.receive()
        if incoming is not None:
            display.show(incoming)
            print(incoming)
        sleep(500)
```

Wenn du die eingehende Nachricht ausgibst, wirst du sehen, dass sie manchmal den Wert `None` enthält. Das liegt daran, dass der micro:bit manchmal nach einer Nachricht sucht, aber noch nichts angekommen ist. Wir können diese `None`-Ereignisse ignorieren, indem wir prüfen, ob `incoming` gleich `None` ist und dann ganz einfach nichts ausgeben.

12.1.4 Verbindung zum Smartphone

Mit Hilfe der microbit Bluetooth Antenne ist es möglich, deinen micro:bit mit deinem Telefon zu verbinden und drahtlos mit dem micro:bit zu kommunizieren. Allerdings unterstützt MicroPython diese Fähigkeit aufgrund mangelnder RAM-Kapazität nicht.

12.2 Übungsaufgaben

- Sende jedes Mal eine Nachricht, wenn die Taste `A` gedrückt wird.
- Du benötigst für diese Aufgabe 2 micro:bits. Programmiere einen micro:bit, um Nachrichten zu empfangen und gib die empfangene Nachricht mit der Methode `print()` aus. Lass diesen micro:bit mit einem USB-Kabel an deinem Computer angeschlossen. Programmiere den andere micro:bit so, dass er die Beschleunigungsmesserwerte oder die Temperaturmesswerte jede Sekunde als Nachricht sendet. Trenne diesen micro:bit vom Computer und benutze eine Batterie, um ihn zu betreiben. Glückwunsch! Du hast einen Datenlogger erstellt!

KAPITEL 13

Ein-/Ausgänge (E/A)

An der Unterkante des BBC micro:bit befinden sich Metallstreifen, die den Eindruck erwecken, dass das Gerät Zähne hat. Das sind die Eingabe- und Ausgabe-Pins (oder kurz E/A Pins). Man nennt sie auch Input/Output bzw. I/O Pins.

13.1 Platinenstecker

Um den micro:bit und seine Funktionen vollständig zu nutzen, um mit der Welt zu kommunizieren, musst du etwas über seinen Platinenstecker lernen. Dieser kann verwendet werden, um sich mit externen Schaltungen und Komponenten zu verbinden.

Abb. 1: Source: https://tech.microbit.org/hardware/edgeconnector/

Einige der Stifte sind größer als andere, so dass es möglich ist, Krokodilklemmen an ihnen zu befestigen. Das sind die mit 0, 1, 2, 3V und GND bezeichneten Pins (Computer beginnen immer bei Null zu zählen). Wenn du ein sogenanntes Edge-Connector- oder BreakOut-Board am Gerät anschließt, kannst du auch ganz leicht Kabel an die anderen (kleineren) Pins anschließen. Die kleineren Pins ermöglichen dir Verbindungen zu den verschiedenen Bauteilen des micro:bit oder du kannst sie für deine eigenen Zwecke verwenden.

Auf dem neuesten micro:bit **V2** kann auch das micro:bit Logo als Touch-Eingang verwendet werden.

Die Beschreibung der einzelnen Pins und wofür sie verwendet werden können, findest du unter <https://microbit.pinout.xyz/>. Siehe auch die *Entwicklerreferenz* für weitere Informationen.

Entwicklerreferenz: https://tech.microbit.org/hardware/edgeconnector/

13.2 Verwendung der Pins

Jeder Pin auf dem BBC micro:bit wird durch ein *Objekt* namens `pinN` repräsentiert wobei `N` die Nummer des Pins ist. Um also zum Beispiel mit dem Pin etwas zu tun der mit einer 0 (Null) beschriftet ist, musst du in deinem Skript das Objekt namens `pin0` benutzen. Der Logo-Pin des **V2** verwendet `pin_logo`.

Diese Pin-Objekte haben verschiedene *Methoden*, die mit ihnen verbunden sind, je nachdem, was der spezifische Pin kann.

13.2.1 Kitzeliges Python

Das einfachste Beispiel für eine Eingabe über die Pins ist eine Überprüfung, ob sie berührt werden. Du kannst also deinen micro:bit kitzeln, um ihn zum Lachen zu bringen! Und das geht so:

```
from microbit import *

while True:
    if pin0.is_touched():
        display.show(Image.HAPPY)
    else:
        display.show(Image.SAD)
```

Halte deinen micro:bit mit einer Hand am GND-Pin. Dann berührst (oder kitzelst) du mit deiner anderen Hand den 0 (Null) Pin. Du solltest sehen, wie sich das Display von grantig zu glücklich verändert!

Wenn du den neuesten micro:bit **V2** verwendest, kannst du auch das Standardverhalten des Pins ändern, so dass du GND gar nicht mehr berühren musst.:

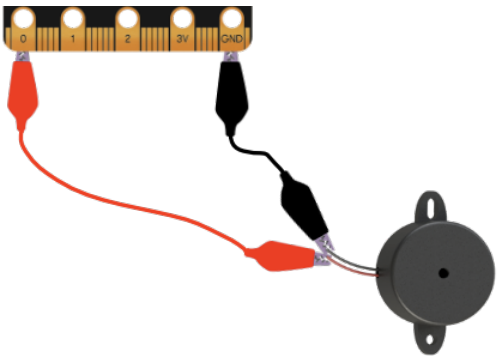
```
from microbit import *
pin0.set_touch_mode(pin0.CAPACITIVE)
while True:
    if pin0.is_touched():
        display.show(Image.HAPPY)
    else:
        display.show(Image.SAD)
```

Dabei handelt es sich um eine sehr einfache Messung eines Eingangs. Der Spaß fängt aber erst richtig an, wenn du Schaltungen und andere Geräte über die Pins ansteckst.

13.2.2 Piepsen und Quietschen

Das Einfachste, das wir an unserem micro:bit anschließen können, ist ein Piezo-Summer. Wir werden ihn für den Output verwenden, dh über ihn werden Töne abgespielt (oder „ausgegeben“).

Diese kleinen Geräte spielen einen hohen Piepton, wenn sie an einen Stromkreis angeschlossen werden. Um so einen Summer an deinen micro:bit anzuschließen, musst du Krokodilklemmen an Pin 0 und GND befestigen, wie das unten gezeigt wird.



Das Kabel von Pin 0 sollte an den (meist längeren) positiven Anschluss des Summers angeschlossen werden.

Das folgende Programm erzeugt am Summer (Buzzer) einen Ton:

```
from microbit import *

pin0.write_digital(1)
```

Das macht etwa 5 Sekunden lang Spaß und dann willst du nur noch, dass das schreckliche Quietschen aufhört. Verbessern wir unser Beispiel und lassen das Gerät sinnvoller piepsen:

```
from microbit import *

while True:
    pin0.write_digital(1)
    sleep(20)
    pin0.write_digital(0)
    sleep(480)
```

Kommst du drauf, was dieses Skript macht? Denke daran, dass 1 in der digitalen Welt „an“ und `0` „aus“ bedeutet.

Das Gerät wird in eine Endlosschleife versetzt und schaltet sofort den Pin 0 ein. Das bewirkt, dass der Buzzer einen Piepton abgibt. Während der Buzzer piepst, macht das Gerät für 20 Millisekunden Pause (es „schläft“) und schaltet dann Pin 0 aus. Das ergibt den Effekt eines kurzen Piepsens. Danach macht das Gerät eine 480 Millisekunden Pause, bevor an den Beginn der Schleife zurückgesprungen wird und alles wieder von vorne anfängt. Das bedeutet, dass du zwei Pieptöne pro Sekunde (einer alle 500 Millisekunden) hörst.

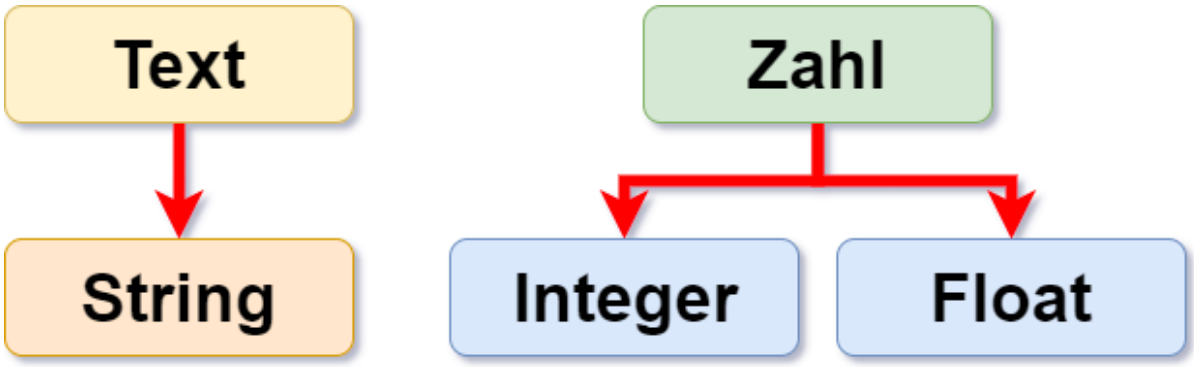
Wir haben ein sehr einfaches Metronom gebaut!

KAPITEL 14

Datentypen

Um verschiedene Arten von Daten korrekt zu erfassen, stellen uns Programmiersprachen verschiedene Datentypen zur Verfügung, damit wir diese richtig darstellen und mit ihnen arbeiten können. Jeder Fall, sei es das Sammeln von Beschleunigungswerten vom Beschleunigungsmesser, das Zählen, wie oft eine Taste gedrückt wurde, oder das Speichern eines Namens, muss auf jeweils ganz spezielle Art und Weise behandelt werden. Deshalb stellt Python so wie die meisten anderen Programmiersprachen mehrere Datentypen zur Darstellung von Werten zur Verfügung.

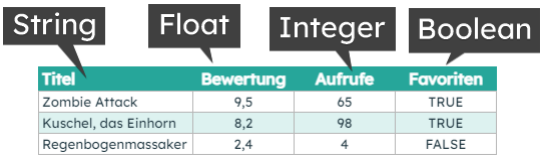
Die Wichtigsten sind:



Eine kleine Übersicht mit Beispielen:

Datentyp	Beschreibung	Beispiel
Integer	Ganze Zahlen	`42`
Float	Zahlen mit dem Dezimalpunkt (kein Komma!), Brüche	`3.1415`
Complex number	Zahlen mit einem Real und einem Imaginärteil	`1 + 3j`
String	Durch Anführungszeichen begrenzte Zeichenketten	`"Hallo Welt!"`
Boolean	Werte, die Wahr oder Falsch sein können	`False`

Auch in einem einfachen Programm wirst du die meisten dieser Datentypen verwenden. Hier sind zB. die Datentypen, die du in einem deiner Programme verwenden könntest, um Informationen über eines deiner micro:bit Spiele zu speichern:



14.1 Operationen

Jeder Datentyp unterstützt unterschiedliche Operationen. Wie wichtig es ist, dass Python weiß um welchen Datentypen es sich handelt, kann man zB an dem uns aus der Mathematik bekannten +-Operator erkennen, der, wie du gleich sehen wirst, ganz unterschiedliche Ergebnisse liefert, je nachdem ob Zahlen oder Strings „zusammengezählt" werden.

14.1.1 Zahlen

Grundlegende arithmetische Operatoren: +, -, * und / werden auf die gleiche Art und Weise wie bei einem Taschenrechner verwendet. Wenn du zur REPL-Konsole wechselst, könntest du beispielsweise folgende Rechnungen nach der Eingabeaufforderung >>> oder, wenn du mehrere Zeilen verwendest, nach ... eintippen und berechnen lassen:

```
>>> 2 + 2
4
>>> # Dies ist ein Kommentar
... 2 + 2
4
>>> 2 + 2  # und dies ist ein Kommentar in derselben Zeile wie Code
4
>>> (50 - 5 * 6) / 4
5.0
>>> 8 / 5 # Brüche gehen nicht verloren, wenn man Ganzzahlen teilt
1.6
```

Schauen wir uns ein Beispiel an, bei dem mit den arithmetischen Operatoren die vom micro:bit gelesene Temperatur von Celsius in Fahrenheit umgerechnet wird:

```
celsiusTemp = temperature()
fahrenheitTemp = celsiusTemp * 9 / 5 + 32
```

Warnung: Python kennt zwei Divisionsoperatoren: / und //. Der erste gibt das Ergebnis aus, das du erwarten würdest, aber der zweite macht eine ganzzahlige Division: der Rückgabewert ist das Ergebnis ohne Rest. Das bedeutet, dass der Rückgabewert immer nach unten gerundet wird.

Um eine Ganzzahldivision auszuführen, die ein ganzzahliges Ergebnis liefert und den Bruchteil des Ergebnisses vernachlässigt, müsstest du also den Operator // anstatt / verwenden

```
>>> # Ganzzahldivision gibt ein abgerundetes Ergebnis zurück:
... 7 // 3
2
>>> 7 // -3
-3
```

Der Operator %, genannt mod wird benutzt, um den Rest zu berechnen, wenn ein Wert durch einen anderen geteilt wird. Zum Beispiel: wenn du wissen willst, ob eine Zahl ungerade oder gerade ist, könntest du versuchen, sie durch 2 zu dividieren. Wenn sie gerade ist, dann gibt es keinen Rest:

```
zahl = 3
if zahl % 2 == 0:
   print("Die Zahl ist gerade")
else:
   print("Die Zahl ist ungerade")
```

Wenn der Rest gleich 1 ist, dann wird dieses Programm `Die Zahl ist ungerade` ausgeben, andernfalls wird es `Die Zahl ist gerade` ausgeben. Du könntest dieses Programm auf eine andere Weise schreiben. Menschen denken über Aufgaben auf unterschiedliche Art und Weise und keine zwei Programme werden wahrscheinlich gleich sein.

14.1.2 Strings

Wie bereits erwähnt, sind Strings (`str` Typ in Python) Zeichenketten, deren Länge nur durch den Speicher deines Rechners begrenzt ist. Ein nützlicher Hinweis ist, dass sie mit dem + Symbol verkettet werden können!

```
name = "Hugo"
nachricht = "Gut gemacht " + name + ". Du hast gewonnen!"
print(nachricht)
```

In der letzten Zeile werden die Elemente auf der rechten Seite von = miteinander verkettet und das Ergebnis in die Variable namens `nachricht` geschrieben. Wenn du diese Zeilen auf der REPL-Konsole eingibst, sollte das so aussehen:

```
>>> name = "Hugo"
>>> nachricht= "Gut gemacht " + name + ". Du hast gewonnen!"
>>> print(nachricht)
Gut gemacht Hugo. Du hast gewonnen!
>>>
```

Was wird ausgegeben, wenn du die folgenden Anweisungen eingibst?

```
>>> a = "1"
>>> b = "2"
>>> summe = a + b
>>> print(summe)
```

Hier kannst du sehr gut sehen, dass es für Python einen großen Unterschied macht ob ich Zahlen als Zahlen oder als Strings abspeichere und verwende. Hier sind `a` und `b` Strings und werden dementsprechend aneinandergefügt und nicht addiert!

Um Zahlen und Strings miteinander zu verbinden, musst du, damit Python nicht durcheinanderkommt, zuerst die Zahl mit der Funktion `str()` in einen String umwandeln:

```
x = temperature
if temperature < 6:
   display.scroll("Kalt" + str(temperature))
```

Bemerkung: Python stellt von Haus aus eine Menge Methoden zur Verfügung, was den Umgang mit Strings sehr vereinfacht und viel Zeit spart (auch wenn die eigene Umsetzung anfangs eine gute Programmierübung sein kann).

14.1.3 Booleans

Ein boolescher Wert (boolean bzw. `bool`) ist ein Wert, der entweder `True` oder `False` ist, auch dargestellt durch *1* und *0*. In Python gibt es eine Reihe von Operationen, die es dir erlauben, boolesche Ausdrücke zu erzeugen.

Vergleiche

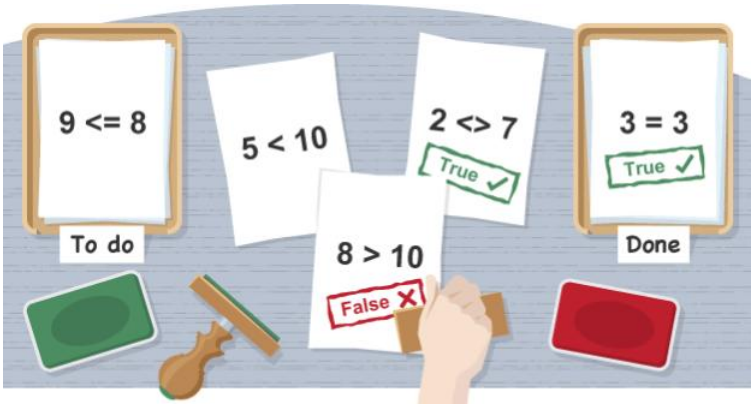


Abb. 1: Source: <http://www.bbc.co.uk/education/guides/zy9thyc/revision>

Vergleichsoperationen sind nützlich, um Variablenwerte in bedingten Anweisungen oder Schleifen zu testen. Hier sind einige Beispiele für Vergleiche, wie sie im Deutschen geschrieben werden:

```
der Punktestand ist größer als 100
der Name ist gleich "Hugo"
die Geschwindigkeit ist ungleich 0
```

Python hat eine Reihe von Vergleichsoperatoren, die es uns erlauben, Vergleiche einfach zu schreiben:

Vergleichsoperator	Bedeutung
==	ist gleich
<, <=	Kleiner als, kleiner als oder gleich
>, >=	Größer als, größer als oder gleich
!=	ist nicht gleich, ungleich

Das Umschreiben der obigen Vergleiche in Python würde lauten:

```
punktestand > 100
name ==  "Hugo"
geschwindigkeit  != 0
```

Logische Operationen

Logische Operatoren testen den Wahrheitswert ihrer Operanden.

Operator	Gibt ``True`` zurück, wenn	Example
and	beide Operanden Wahr sind	`True and True`
or	Mindestens ein Operand Wahr ist	`True or False`
not	der Operand Falsch ist	`not False`

Zugehörigkeitsoperatoren

Zugehörigkeitsoperatoren sind nützlich, um das Vorhandensein eines Elements in einer Sequenz zu bestimmen.

Operator	Gibt ``True`` zurück, wenn	Beispiel
in	sich ein Variablenwert in der angegebenen Reihe befindet	`x in [1, 2, 3, 4]`
not in	kein Variablenwert in der angegebenen Liste gefunden wird	`x not in [1, 2, 3, 4]`

Boolesche Operationen verwenden

Du hast vielleicht schon einige Beispiele verwendet, die so etwas machen. In diesem Beispiel wird der micro:bit einen Pfeil anzeigen, der seine Richtung entsprechend der Beschleunigung ändert:

```
from microbit import *

while True:
    x_richtung = accelerometer.get_x()

    if (x_richtung <= 100) and (x_richtung >= 50):
        display.show(Image.ARROW_N)

    elif x_richtung > 100:
        display.show(Image.ARROW_E)

    elif  x_richtung < 50:
        display.show(Image.ARROW_W)

    else:
        display.show(Image.ARROW_S)
```

14.2 Übungsfragen

1. **Gib an, ob der Rückgabewert True oder False ist. Wenn False, erkläre warum.**
   Überprüfe danach deine Vermutung auf der REPL-Konsole.

   a) `"hello" == 'hello'`

   b) `10 == 10.0`

   c) `5/2 == 5//2`

   d) `5 in [x for x in range(0,5)]`

   e) `0 == False`

   f) `1 == true`

   g) `0.1 + 0.2 == 0.3`

KAPITEL 15

Variablen

Eine Variable ist ein Name, den du verwendest, um auf einen Speicherplatz zu verweisen, an dem ein Wert gespeichert ist. Einfacher ausgedrückt kann man sie sich als eine Box vorstellen, die einen Wert speichert. Alle Variablen bestehen aus drei Teilen: einem Namen, einem Datentyp und einem Wert. In der Abbildung unten gibt es drei Variablen mit unterschiedlichen Datentypen:

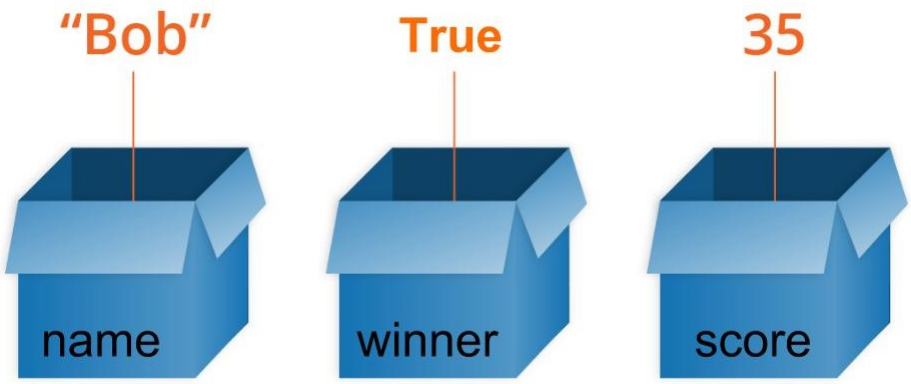


Abb. 1: Source: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables>

Die Variable `name` enthält die Zeichenkette `Bob`, die Variable `winner` enthält den Wert `True` und die Variable `score` enthält den Wert `35`.

In Python wird eine Variable erstellt, wenn ihr zum ersten Mal ein Wert zugewiesen wird. Wenn du eine Variable `n` im Code verwendest, der du vorher keinen Wert zugewiesen hast, kommt es zu folgender Fehlermeldung:

```
>>> # Versuche eine undefinierte Variable abzurufen
... n
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Sobald einer Variablen ein Wert zugewiesen wurde, kann sie verwendet und dieser Wert auch verändert werden (es sei denn, sie ist unveränderlich - mehr darüber erfährst du im Abschnitt Datenstrukturen).

```
from microbit import *

meineAnzahl = 0

while True:
        if button_a.was_pressed():
                meineAnzahl = meineAnzahl + 1
                print("Die Taste wurde " + str(meineAnzahl) + " mal gedrückt.")  #␣
↪Ausgabe auf der REPL
                        sleep(2000)
```

Hier haben wir die Variable `meineAnzahl` benutzt, um mitzuzählen, wie oft die taste `A` gedrückt wird. Kannst du sagen, was dieser Codeschnipsel sonst noch macht?

Die `sleep()`-Funktion werden wir oft verwenden, da in einer Endlosschleife aufgrund der Verarbeitungsgeschwindigkeit des Prozessors oft unerwartete Ergebnisse zu bewundern sind. Um sie richtig einzusetzen, musst du dein Programm so planen, dass die Verzögerungen, die durch den Aufruf dieser Funktion ausgelöst werden, zu der jeweils richtigen Zeit passieren. Das musst du genau planen!

15.1 Übungsaufgaben

1. Was passiert, wenn wir die `sleep()`-Funktion weglassen?
2. Gib in der REPL nacheinander folgende Befehle ein: a=1 ; b=2; print(a+b); a=‘1‘ ; b=‘2‘ ; print(a+b) . Erkläre die unterschiedlichen Ergebnisse.

KAPITEL 16

Kontrollstrukturen

Das erste Programm im Abschnitt *Erste Schritte* bestand aus *aufeinanderfolgenden Anweisungen*, die einfach nur *nacheinander ausgeführt* wurden. In den meisten Fällen wirst du jedoch eine komplexere Struktur für deinen Code benötigen. Du wirst dabei kontrollieren wollen, welche Anweisungen ausgeführt werden oder wie oft sie ausgeführt werden. Dies ist der Zeitpunkt, an dem die in diesem Abschnitt vorgestellten Kontrollstrukturen - wie z.B. *Schleifen* oder *bedingte Anweisungen* - nützlich sind.

16.1 Bedingte Anweisungen mit if

Das erste Beispiel für einen Anwendungsfall von Kontrollstrukturen liegt dann vor, wenn du einen Teil deines Codes nur dann ausführen willst, **wenn** (`if`) eine bestimmte Bedingung erfüllt ist. Zum Beispiel, wenn du ein Ereignis nur auslösen willst, wenn eine Taste gedrückt wird (`if button_a.is_pressed()`):

```
from microbit import *
import love

while True:
    if button_a.is_pressed():
        love.badaboom()
    elif button_b.is_pressed():
        display.show(Image.HAPPY)
    else:
        display.show(Image.GHOST)
    sleep(100)
```

Falls du eine andere Aufgabe unter verschiedenen Bedingungen ausführen willst, verwende die Anweisung `elif` (kurz für `else if`).

Die `else`-Anweisung ist nützlich, wenn du etwas dann tun willst, wenn keine Bedingung zutrifft. Die beiden letztgenannten Anweisungen sind nur verwendbar, wenn du zuvor eine `if`-Anweisung verwendet hast, aber keine davon ist zwingend notwendig.

Ein einfaches Beispiel dazu findest du im Kapitel Tasten bei der Programmierung eines Tamagotchi.

16.2 Schleifen (Loops)

Schleifen sind eine sehr nützliche Struktur, wenn du einen bestimmten Codeblock mehrmals wiederholen willst. Es gibt zwei Arten von Schleifen: `for`-Schleifen, die mitzählen, wie oft ein Codeblock ausgeführt wird, und `while`-Schleifen, die eine Aktion ausführen, bis eine Bedingung, die du angegeben hast, nicht mehr wahr ist.

16.2.1 While Schleife

Ereignisschleifen

Oft muss dein Programm einfach nur drauf warten, dass etwas passiert. Um dies zu erreichen, lässt du es einen Code, der definiert, wie es auf bestimmte erwartete Ereignisse, wie zum Beispiel das Drücken einer Taste, reagieren soll, andauernd wiederholen. Der Code durchläuft also eine Schleife und bricht erst ab, wenn das erwartete Ereignis eingetreten ist.

Um Schleifen in Python zu programmieren, benutzt du das `while` Schlüsselwort. Der Code prüft, ob eine Bedingung `True` ist. Wenn ja, wird ein *Codeblock* ausgeführt, der *Body* der Schleife genannt wird. Wenn nicht, bricht das Programm aus der Schleife aus (ignoriert den Body) und macht direkt nach dem Block weiter.

Python macht es einfach, Codeblöcke zu definieren. Sagen wir, ich habe eine To-Do-Liste auf einen Zettel geschrieben. Sie könnte etwa so aussehen:

```
Einkaufen
Kaputte Dachrinne reparieren
Den Rasen mähen
```

Wenn ich meine To-Do-Liste noch weiter unterteilen möchte, würde ich vielleicht etwas schreiben wie das hier:

```
Einkaufen:
    Eier
    Speck
    Tomaten
Kaputte Dachrinne reparieren:
    Leiter von nebenan leihen
    Hammer und Nägel finden
    Leiter zurückbringen
Den Rasen mähen:
    Rasen um Teich auf Frösche kontrollieren
    Kraftstoffstand des Rasenmähers prüfen
```

Es ist offensichtlich, dass die Hauptaufgaben in Unteraufgaben aufgeteilt sind, die *Einrückung* unterhalb der Hauptaufgabe, zu der sie gehören. Also `Eier`, `Speck` und `Tomaten` offensichtlich mit `Einkaufen` verwandt sind. Durch das Einrücken von Dingen können wir auf einen Blick erkennen, wie die Aufgaben miteinander zusammenhängen.

Dies wird *Verschachtelung* genannt. Wir benutzen die Verschachtelung, um Codeblöcke wie diesen zu erstellen:

```
from microbit import *

while running_time() < 10000:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    display.show(Image.ASLEEP)

display.show(Image.SURPRISED)
```

Die Funktion `running_time()` gibt die Anzahl der Millisekunden zurück, seit das Gerät gestartet wurde.

Die Zeile `while running_time() < 10000:` prüft, ob die laufende Zeit weniger als als 10000 Millisekunden (d.h. 10 Sekunden) ist. Wenn ja wird das Display `Image.ASLEEP` anzeigen. Beachte wie diese Zeile unter der `While`-Anweisung eingerückt ist, genau wie in unserer To-Do-Liste.

Offensichtlich, wenn die Laufzeit gleich oder größer als 10000 Millisekunden ist dann wird auf dem Display `Image.SURPRISED` angezeigt. Warum? Weil die `while` Bedingung falsch sein wird (Die Laufzeit `running_time()` ist nicht mehr < `10000`). In diesem Fall ist die Schleife beendet und das Programm fährt nach der `while` Schleife mit dem Code-Block fort. Es wird dann so aussehen, als würde dein Gerät für 10 Sekunden schlafen, bevor es mit einem überraschten Blick im Gesicht wieder aufwacht.

Probiere es aus!

Aber was ist, wenn du eine Aktion nur ausführen willst, während etwas passiert? Vielleicht möchtest du ein Bild anzeigen wenn die Temperatur auf dem micro:bit unter einen bestimmten Wert fällt, also musst du die Temperatur prüfen:

```
from microbit import *

while (temperature() < 18):
    display.scroll(Image.SAD)
    sleep(1000)

display.show(Image.HAPPY)
```

Endlosschleife

Eines der häufigsten Dinge, die du mit einer `while` Schleife machen kannst, ist, etwas für immer ablaufen zu lassen, d.h. bis der micro:bit ausgeschaltet oder zurückgesetzt wird. Das benötigst du, damit der micro:bit zB in einem Spiel auf deine Eingaben wartet und darauf reagieren kann. Oder du lässt ihn andauernd die Temperatur überwachen und mitschreiben.

Hier ist ein Beispiel für einen Code, der sich ewig wiederholt:

```
from microbit import *

while True:
    display.scroll("Hallo Welt")
```

Dieser Code wird wiederholt die Meldung `Hallo Welt` anzeigen. Du wirst wahrscheinlich immer mindestens eine `while True:` Schleife in deinem Programm einsetzen, um den micro:bit am Laufen zu halten.

16.2.2 For Schleife

Es kommt vor, dass du eine Aktion eine bestimmte Anzahl von Malen ausführen willst, oder du musst nachverfolgen, wie oft sie ausgeführt wurde. Zum Beispiel möchtest du die LEDs auf der obersten horizontalen und der rechten vertikalen Seite anschalten. Du kannst eine `for` Schleife verwenden, um zu ändern, welche LED leuchtet.:

```
from microbit import *

for i in range(5):  # range(5) entspricht der Liste [0,1,2,3,4], hat also 5 Werte
   # Setze das Pixel in der Spalte 0, Zeile i auf 9
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    display.set_pixel(0, i, 9)

    # Setze das Pixel in der Spalte 4, Zeile i auf 9
    display.set_pixel(4, i, 9)
```

Hier ist ein weiteres Beispiel. Du könntest eine `for Schleife` verwenden, um alle LEDs nacheinander einzuschalten, eine nach der anderen:

```
from microbit import *

display.clear()
for x in range(0, 5):
    for y in range(0, 5):
        display.set_pixel(x, y, 9)
        sleep(100)
```

Die `for Schleife` lässt dich eine Schleife mit Hilfe eines Zählers eine bestimmte Anzahl von Malen ausführen. Die äußere Schleife:

```
for x in range(0,5):
```

führt die Schleife fünfmal aus und ersetzt jedes Mal `x` durch aufeinanderfolgende Werte im Bereich `0` bis `4` (in Python und den meisten Programmiersprachen, beginnen wir immer von `0` zu zählen). Die Schleife hört auf, bevor sie 5, den letzten Wert im Bereich, erreicht.

Die innere Schleife:

```
for y in range(0,5):
```

führt die Schleife fünfmal aus und ersetzt `y` jedes Mal durch aufeinanderfolgende Werte im Bereich `0` bis `4`. Auch hier stoppt die Schleife, bevor sie den letzten Wert im Bereich erreicht.

Was glaubst du, macht das folgende Programm?

```
from microbit import *
import random

display.clear()

while True:
    for x in range(5):
        for y in range(5):
            display.set_pixel(x, y, random.randrange(10))
            sleep(100)
```

16.3 Übungsaufgaben

1. Zeige ein anderes Bild an, je nachdem in welche Seite der microbit gekippt ist.
2. Programmiere ein LED ‚Symbol', das sich in die Richtung bewegt, in die der micro:bit gekippt ist.
3. Programmiere ein Programm, das bis 5 zählt und jede Zahl 500ms anzeigt.
4. Programmiere einen Countdown der von 3 herunterzählt und jede Zahl 1s anzeigt.
5. Die Funktion `display.scroll()` kann einen Text auch ohne while-Schleife als andauernde Laufschrift darstellen. Finde heraus wie!

KAPITEL 17

Datenstrukturen

17.1 Liste

Eine Liste („Array“) ist eine Datenstruktur, die verwendet wird, um einen beliebigen Datentyp (oder eine andere Struktur) in geordneter Weise zu speichern. Eine Liste ist die Datenstruktur, die du wahrscheinlich am häufigsten benutzen wirst. Sagen wir, wir wollen den Punktestand eines Spielers speichern. Wir könnten eine Liste wie die oben abgebildete verwenden. Die Liste hat eine „Box“ für jeden Wert. Die Daten, die in einer Liste gespeichert sind, werden „Elemente“ genannt.

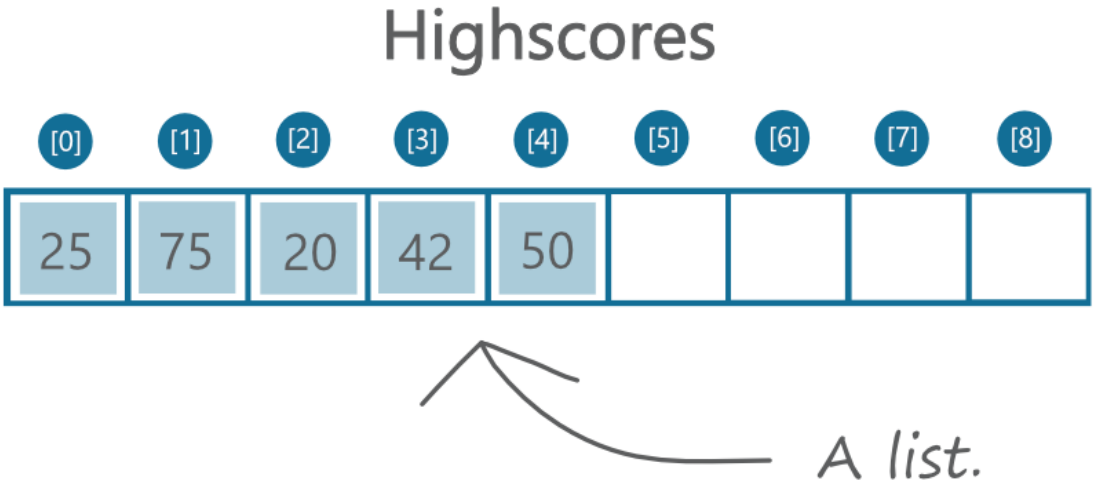


Abb. 1: Eine Liste.

Um eine Liste zu erstellen, gibst du ihren Inhalt in eckigen Klammern an und trennst die einzelnen Elemente durch Kommas:

```
from microbit import *

high_scores = [25, 20, 10, 15, 30]      # Erstelle eine Liste und speichere einige␣
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
↪Werte in ihr.
print(high_scores[0])                    # Gibt 25 aus
print(high_scores[3])                    # Gibt 15 aus
```

Den Wert eines der Elemente in einer Liste zu finden ist einfach, solange du im Hinterkopf behältst, dass Python die Elemente ab ‚0' zählt. In der `high_scores` Liste oben, ist `high_scores[0]` 25 und `high_scores[3]` ist 15.

Hier kannst du auch sehen, dass auf bestimmte Elemente in einer Liste über ihren Index zugegriffen werden kann. Außerdem ist es möglich, Listen zu zerschneiden, um nur einen Teil der Liste abhängig vom Index zu erhalten. Wenn du nur die ersten drei willst, kannst du `high_scores[0:3]` schreiben, oder, da wir bei 0 anfangen, können wir es zu `high_scores[:3]` abkürzen. Beachte, dass der rechte Endpunkt immer ausgeschlossen ist, also bezieht sich der obige „Ausschnitt" auf das mathematische Intervall `[0,2]`.

Natürlich besitzt Python schon Funktionen für die Arbeit mit Listen. Der folgende Codeschnipsel berechnet die Summe aller Elemente und berechnet dann den durchschnittlichen Punktestand.

```
punkte_gesamt = 0

for punkte in high_scores:               # Für jedes Element ...
        punkte_gesamt = punkte_gesamt + punkte

durchschnitt = punkte_gesamt / len(high_scores)  # Die Funktion len() gibt die "Länge
↪" der Liste zurück
```

Das Gleiche kann sogar in einer Zeile mit der Funktion `sum` gemacht werden:

```
durchschnitt_kurzfassung = sum(high_score) / len(high_score)
```

Da du nicht unbedingt weißt, welche Werte in der Liste sein werden, oder wie groß die Liste sein wird, ist es nützlich, die `append` Funktion zu kennen. Du kannst sie zum Beispiel benutzen, um eine Liste mit Temperaturmesswerten oder Beschleunigungsmessungen zu füllen:

```
from microbit import *

temperaturaufzeichnungen = []            # Erstelle eine leere Liste
for i in range(100):                     # 100 Temperaturwerte hinzufügen
        temperaturaufzeichnungen.append(temperature())
        sleep(1000)
```

Die `for` Schleife wird 100 mal ausgeführt und `i` hat Werte von 0 bis 99. Dadurch wird die Temperatur jede Sekunde für 100 Sekunden gemessen und der Wert an das Ende der Liste angefügt.

Das Löschen von Elementen aus einer Liste ist genauso einfach:

```
high_scores.delete(24)
```

Dadurch wird das erste Element mit dem Wert 24 gelöscht. Alternativ kannst du auch ein Element an einer bestimmten Position löschen, wenn du es kennst:

```
high_scores.pop(3)
```

Dies löscht oder ‚poppt' das Element an der angegebenen Position in der Liste. Beachte, dass:

```
high_scores.pop()
```

das letzte Element in der Liste löscht.

Tipp: Du kannst hier nachschauen, um weitere nützliche Methoden für Listen zu sehen.

Bemerkung: Du fragst dich vielleicht, ob Strings als Liste betrachtet werden können. Auch wenn ein String ein Array von Zeichen ist und wir sogar ähnliche Operationen mit ihnen durchführen können (wie z.B. Slicing), sind sie beide unterschiedliche Objekttypen mit unterschiedlichen Methoden (versuche `dir(str)` und `dir(list)` in deiner Konsole einzugeben).

17.1.1 Sortieren

Oft wirst du die Daten in deiner Liste sortiert haben wollen, zum Beispiel bei der Implementierung von Suchalgorithmen. In Python ist es einfach, Listen zu sortieren, indem man die `sort()` Funktion verwendet:

```
high_scores = [25, 20, 10, 15, 30]
high_scores.sort()
```

Du kannst nicht nur Zahlen sortieren - der optionale Parameter `key=` erlaubt es dir, deine eigene Funktion für den Vergleich von Elementen in deiner Liste anzugeben (zum Beispiel, wenn du eine Liste von Strings nach der Länge sortieren willst, kannst du die Funktion len() als Parameter übergeben). Die Übergabe von false an den Parameter `reverse=` erlaubt es dir, in absteigender Reihenfolge zu sortieren.

```
list = ['länger', 'kurz', 'am längsten']

# Liste in aufsteigender Reihenfolge der Stringlänge sortieren
list.sort(key=len)
# Liste in absteigender Reihenfolge der Stringlänge sortieren
list.sort(key=len, reverse=True)
```

17.2 Tupel

Tupel sind ähnlich wie Listen, da sie verwendet werden, um eine geordnete Folge von Elementen zu speichern, die normalerweise einen unterschiedlichen Datentyp haben.:

```
high_scores_unveraenderlich = (25, 20, 10, 15, 30)
```

Du kannst Werte auf die gleiche Weise wie mit Listen abrufen, aber der wichtigste Unterschied ist, dass Tupel *unveränderlich* sind. Das bedeutet, dass du in der `high_scores` Liste oben, den Wert einzelner Elemente ändern kannst:

```
high_scores[0] = 42
```

Der Versuch, einen Wert innerhalb von `high_scores_unveraenderlich` zu ändern, gibt jedoch einen `TypeError: Object tuple does not support item assignment` zurück. Sobald du Werte innerhalb eines Tupels zugewiesen hast, können sie nicht mehr geändert werden.

Die Veränderbarkeit ist ein weiterer Unterschied zwischen Strings und Listen - während Listen veränderbar sind, sind es Strings nicht.

17.3 Set

Im Gegensatz zu Listen und Tupeln, enthalten Sets eine **ungeordnete** Sammlung von Elementen ohne Duplikate. Das ermöglicht das Testen der Zugehörigkeit oder das Entfernen doppelter Elemente.

```
set = {8, 12, 22}

# Ein einzelnes Element zum Set hinzufügen.
set.add(42)

# Mehrere Elemente zum Set hinzufügen
set.update([16, 32, 64])

# Entferne ein Element aus dem Set - gibt einen Fehler aus, wenn das Element nicht im␣
↪Set ist
set.remove(42)

# Entferne ein Element wenn es im Set vorhanden ist
set.discard(42)
```

Da ein Set eine ungeordnete Sammlung von Elementen ist, ist eine Indexierung nicht möglich. Python unterstützt typische Set-Operationsmethoden:

```
set_a = {1,2,3,4,5}
set_b = {4,5,6,7}
set_c = {1,2}

# Überprüfung auf Zugehörigkeit
2 in set_a

# Gib Elemente in der Schnittmenge von set_a und set_b zurück
set_a.intersection(set_b)
# Gib true zurück, wenn set_a alle Elemente von set_c enthält
set_a.issuperset(set_c)
```

Ein leeres Set wird mit der Methode `set()` erzeugt, da die Verwendung von geschweiften Klammern ein leeres Dictionary erzeugt (siehe unten).

Für weitere Methoden, siehe Python Dokumentation.

17.4 Dictionary

Ein Dictionary ist ein ungeordnetes Set von `Schlüssel : Wert` Paaren. Es ist eine Regel, dass alle Schlüssel eindeutig sind und keine Duplikate haben. Anders als Listen oder Tupel, die durch Zahlen indiziert werden, kannst du einen Wert aus einem Dictionary abrufen, indem du den Schlüssel als Index verwendest.

Hier ein Vergleich zwischen `List` und `Dictionary`:

Zum Beispiel kannst du die Highscores aller Spieler auf diese Weise speichern:

```
game_register = { 'googolplex': 100,
                  'terminat0r': 27,
                  'r00t': 150,
                  'dent': 42,
                  'teapot418' : 0 }

# auf Elemente zugreifen
```

(Fortsetzung auf der nächsten Seite)

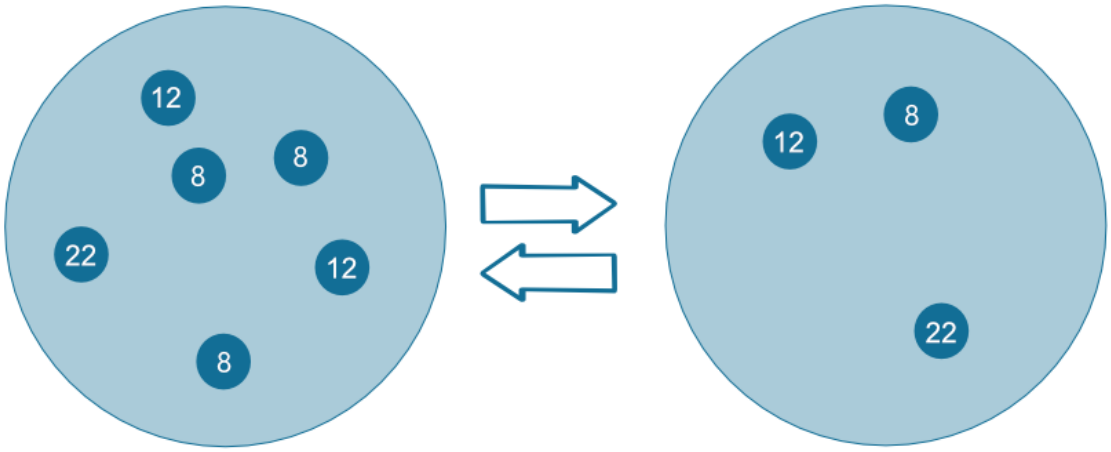
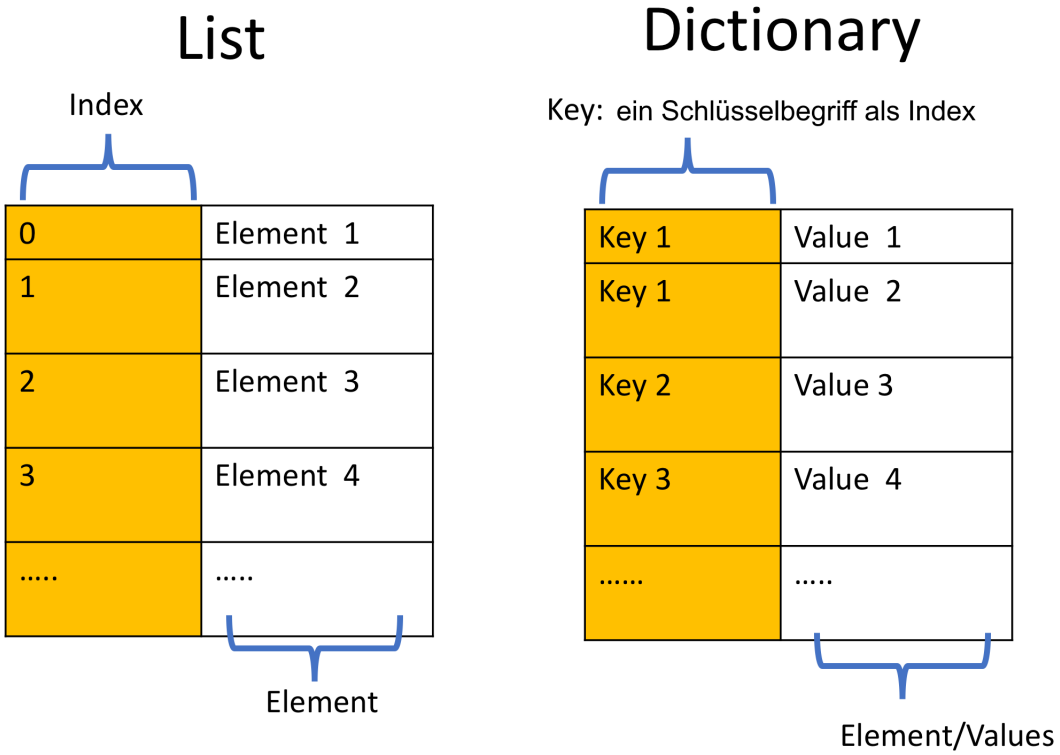


Abb. 2: Alle Elemente innerhalb eines Sets sind einzigartig



(Fortsetzung der vorherigen Seite)

```
game_register['dent']

# Hinzufügen oder aktualisieren eines bestehenden Eintrags
game_register['pepper'] = 50

# Einen Eintrag löschen
del game_register['pepper']

# Alle Einträge löschen
game_register.clear()

# Das Dictionary löschen
del game_register

# Rufe einen Wert für den Schlüssel oder den Standardwert ab, wenn er nicht im␣
↪Dictionary steht
game_register.get('dent')
```

17.5 Übungsaufgaben

1. Verwende die micro:bit Liste `Image.ALL_CLOCKS` und durchlaufe alle Elemente in der Liste mit einer for Schleife und zeige sie auf dem LED Bildschirm an.
2. Verwende dieselbe Elementeliste und zeige nur Elemente mit einem durch 3 teilbaren Index.
3. Sortiere eine Integer-Liste (z.B. `Liste = [20, 112, 45, 80, 23]`) anhand der letzten Ziffer jedes Elements und behalte ihre relativen Positionen bei, falls die Ziffer gleich ist (das Ergebnis wäre in diesem Fall `[20, 80, 112, 23, 45]`).
4. Erstelle eine eigene Animation mit einem Tupel und spiele sie auf dem micro:bit LED Bildschirm ab.
5. Programmiere den microbit so, dass er beim Drücken einer Taste eine Kompassmessung vornimmt und die Ergebnisse in einem Tupel speichert.
6. Schreibe ein Programm, das die Gesten, die der microbit erkennt, und die Anzahl ihrer Erkennung in einem Dictionary festhält.

KAPITEL 18

Funktionen I

Funktionen und Methoden beziehen sich auf nützliche Codeschnipsel, die einem bestimmten Zweck dienen und normalerweise viele Male in deinem Programm verwendet werden. Wahrscheinlich hast du sowohl Funktionen als auch Methoden schon benutzt, ohne es unbedingt zu merken. In diesem Abschnitt werden wir nicht weiter auf Methoden eingehen, aber wir werden erklären, wie man Funktionen benutzt und schreibt.

18.1 Funktionen verwenden

Eine großartige Sache an Funktionen ist, dass sie in mehr als einem Programm verwendet werden können und Code-Wiederholungen vermeiden. Auf die gleiche Weise kannst du Funktionen verwenden, die andere Leute geschrieben haben. In Python können nützliche Funktionen zu Modulen zusammengestellt werden (obwohl du das nicht tun musst) - das `random` Modul ist ein gutes Beispiel. Um die Funktionen des Random-Moduls zu nutzen, musst du das Modul zuerst *importieren*. Sobald du das getan hast, kannst du alle Funktionen des Moduls benutzen. Hier sind zwei Beispiele von Funktionen aus ‚random'.

Die Funktion `random.randint()` erlaubt es uns, eine zufällige Ganzzahl (Integer) aus einem bestimmten Bereich zu erzeugen:

```
from microbit import *
import random

display.show(str(random.randint(1, 6)))
```

Im obigen Code wird eine Zufallszahl von 1 bis 6 generiert - die obere Grenze ist in diesem Fall (ausnahmsweise!) mit dabei, dh jede der Zahlen 1, 2, 3, 4, 5 und 6 können angezeigt werden.

In diesem Codeschnipsel prüft die Funktion `random.choice` wie viele Elemente in der Namensliste sind, erzeugt eine zufällige Ganzzahl im Bereich von 0 bis zur Listenlänge und gibt ein Listenelement mit dem Index der erzeugten Ganzzahl zurück:

```
from microbit import *
import random

namen = ["Maria", "Jolanda", "Damien", "Alia", "Kushal", "Mei Xiu", "Zoltan" ]

display.scroll(random.choice(namen))
```

18.2 Eigene Funktionen erstellen

Funktionen können dir helfen, deinen Code zu organisieren und ihn ordentlich zu halten. Hier ist ein Beispiel für eine einfache Funktion, die eine Nachricht ausgibt:

```
def zeigeBegruessung():
        print("Hallo alle miteinander!")
```

Um die Funktion zu nutzen, muss sie *aufgerufen* werden:

```
zeigeBegruessung()
```

Oder du könntest in einem Spiel an verschiedenen Stellen einen Countdown verwenden und willst nicht immer dieselbe for-Schleife im Code verwenden sondern einfach nur `countdown()` schreiben.

```
#Countdown 3 - 2 - 1
def countdown():
        for i in range(3,0,-1):   # Werte von i zählen von 3 herunter
        display.show(i)
        sleep(500)
        display.clear()

...
countdown()
...
countdown()
...
```

Das sind keine sehr interessante Funktionen, oder? Du kannst Funktionen mächtiger machen, indem du *Parameter* und *Rückgabewerte* benutzt. Wenn du dir eine Funktion wie eine Black Box vorstellst dann ist ein Parameter ein Eingabewert und ein Rückgabewert ist das, was du am anderen Ende wieder heraus bekommst.

Wir könnten zum Beispiel die Countdown-Funktion erweitern, so dass wir ihr mit einem Parameter, den wir beim Aufrufen übergeben, sagen, von welchem Startwert `n` aus sie beginnen soll herunterzuzählen. Dafür müssen wir in der Klammer der Funktion `countdown()` die Variable n als Parameter der Funktion eintragen. In dieser Variablen `n` wird der Übergabewert gespeichert und kann dann im Code der Funktion verwendet werden.

```
def countdown(n):
        for i in range(n,0,-1):  # Werte von i zählen von n herunter
        display.show(i)
        sleep(500)
        display.clear()
```

Der Parameter `n` macht unsere Funktion also gleich viel flexibler und in unterschiedlichen Situationen einsetzbar.

Oder nehmen wir an, wir wollen ein kleines Programm schreiben, das einige Freunde mit einer Nachricht begrüßt, die ihren Namen und ihr Alter enthält:

```
from microbit import *

def printBirthdayGreeting(name, age):
    return "Happy Birthday " + name + " , du bist unglaubliche " + str(age) + " Jahre␣
↪alt!"


display.scroll(printBirthdayGreeting("Toni", 8))
display.scroll(printBirthdayGreeting("Sonja", 9))
display.scroll(printBirthdayGreeting("Maria", 11))
```

Die Funktion `printBirthdayGreeting` stellt die Geburtstagsnachricht für uns zusammen und gibt einen String zurück. `str()` wird benutzt, um das `Alter`, welches eine Zahl ist, in einen String zu verwandeln.

Du musst keine Parameter oder Rückgabewerte in deinen Funktionen verwenden, es sei denn, du willst/brauchst sie.

18.3 Übungsaufgaben

1. Schreibe eine Funktion `blink(x, y)`, die eine LED an den Koordinaten, die durch die Parameter x und y angegeben werden, einmal blinken lässt.
2. Benutze die Funktion `blink(x, y)` um alle LEDs nacheinander blinken zu lassen.
3. Schreibe eine Funktion button_count() die ein Tupel zurückgibt, das die Anzahl der Betätigungen von Taste A und Taste B enthält. (fix)
4. Kombiniere die beiden Funktionen in einem Programm, das es dem Benutzer ermöglicht, die Koordinaten der blinkenden LED durch Drücken der Tasten einzustellen.
5. Schau dir die Skripte an, die du zuvor geschrieben hast und prüfe, ob du deinen Code durch die Verwendung von Funktionen verbessern könntest (oder nicht).

KAPITEL 19

Funktionen II

Jetzt, wo du weißt, wie man Funktionen in der Praxis einsetzt, gibt es einige weitere Konzepte, die dir helfen werden, das Verhalten von Funktionen nicht nur in Python zu verstehen, sondern auch in anderen Sprachen.

19.1 Geltungsbereich (Scope)

19.1.1 Globale und lokale Variablen

Stell dir vor, du möchtest eine leicht abgeänderte Funktion `printBirthdayGreeting()` von vorher verwenden und du möchtest das Alter bei jedem Aufruf der Funktion erhöhen:

```
name = "Johann"
alter = 32

def printBirthdayGreeting():
        alter += 1
    return "Happy Birthday " + name + ", du bist " + str(alter) + " Jahre alt!"

printBirthdayGreeting()
```

Kannst du erkennen, was falsch ist? Wenn du versuchst, den Code auszuführen, wirst du wahrscheinlich diese Meldung erhalten: UnboundLocalError: local variable ‚alter' referenced before assignment``.

Um dies zu verstehen, müssen wir über den Geltungsbereich (Scope) sprechen. Scope ist eine ‚Umgebung', in der eine Variable definiert ist, und auf die dort zugegriffen und in die geschrieben werden kann. Unter diesem Gesichtspunkt kennen wir zwei Typen von Variablen: globale und lokale.

Standardmäßig sind alle Variablen, die innerhalb einer Funktion definiert sind, lokal - du kannst nicht auf sie außerhalb der Funktion zugreifen. Und da der Geltungsbereich innerhalb der Funktion anders ist als der globale, ist es möglich, den gleichen Namen für zwei verschiedene Variablen zu verwenden.

Kannst du jetzt erklären, was im obigen Codeschnipsel passiert ist?

`alter` außerhalb der `printBirthdayGreeting()` Funktion ist eine globale Variable. Wenn wir jedoch innerhalb der Funktion darauf zugreifen wollen, betrachtet Python es als eine neue lokale Variable. Wie lösen wir das Problem? Wir können die Variable `alter` als `global` deklarieren:

```
name = "Johann"
alter = 32

def printBirthdayGreeting():
        global alter
        alter += 1
        return "Happy Birthday " + name + ", du bist " + str(alter) + " Jahre alt!"
```

Dadurch weiß Python, dass die age-Variable, die wir meinen, im globalen „Namespace“ (wortwörtlich *Namensbereich*) liegt.

Warnung: Globale Variablen zu verwenden ist generell eine schlechte Praxis und du solltest es vermeiden, da es den Zweck deiner Funktionen weniger offensichtlich macht und du am Ende mit ‚Spaghetti‘ Code kämpfen musst. Ein besserer Weg ist es, die Variable `age` als eines der Argumente der Funktion zu übergeben (Beispiel unten).

Hier ist ein Beispiel für eine Funktion, die Variablen als Argumente übergibt:

```
def printBirthdayGreeting(name, alter):
        alter += 1
        return "Happy Birthday " + name + ", du bist " + str(alter) + " Jahre alt!"
```

Tipp: Du wirst oft von „Best Practices“ hören. Wie bestimmst du, was eine Best Practice ist und was nicht? Im Allgemeinen ist eine Best Practice das, was deinen Code für andere besser lesbar macht. Du kannst dir Styleguides für eine Sprache ansehen, in der du programmierst, aber am Ende geht es immer um gutes Urteilsvermögen, da keine Regel in allen Fällen passt.

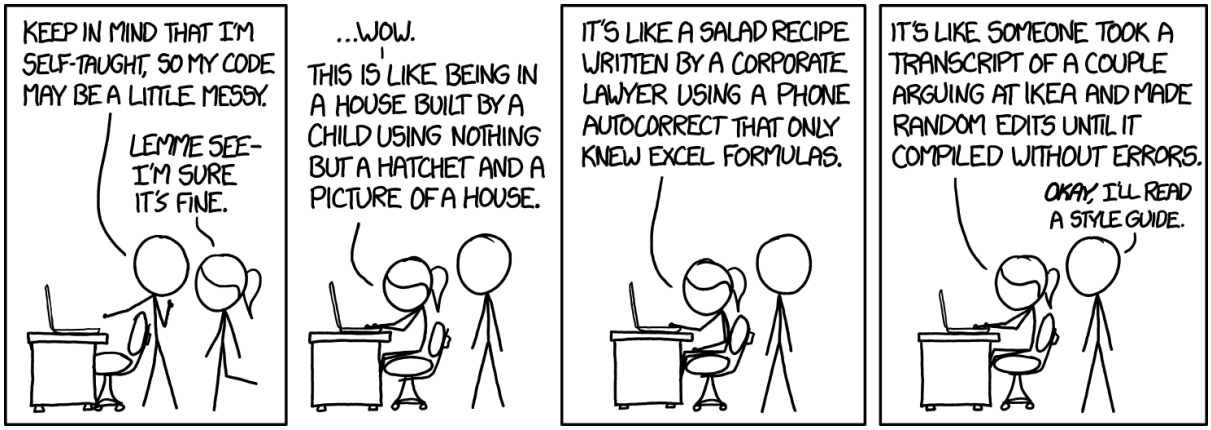


Abb. 1: Quelle: xkcd https://xkcd.com/1513/

19.1.2 Nicht-lokale Variablen

Ein kurioser Fall tritt bei der Verwendung von verschachtelten Funktionen auf. Sagen wir, du willst eine lokale Variable der Funktion `nurEinBeispiel()` ändern, indem du die verschachtelte Funktion:

```
def nurEinBeispiel():
        def weiteresBeispiel():
                variable = "Innere Funktion, die alles verändert!"

        variable = "Äußere Funktion"
        weiteresBeispiel()

        print(variable)

nurEinBeispiel()
```

Du weißt bereits, warum das nicht funktioniert. Aber wie kannst du das Problem umgehen? Du kannst die Variable nicht global deklarieren, weil sie innerhalb einer Funktion ist - sie ist lokal und es gibt einen weiteren lokalen Bereich innerhalb der Funktion `weiteresBeispiel()`. Um diese Situation zu lösen, kannst du eine Variable als `nonlocal` deklarieren:

```
def nurEinBeispiel():
        def weiteresBeispiel():
                nonlocal variable
                variable = "Innere Funktion, die alles verändert!"

        variable = "Äußere Funktion"
        weiteresBeispiel()

        print(variable)

nurEinBeispiel()
```

Jetzt sollte der Code `"Innere Funktion, die alles verändert!"` genau so ausgeben, wie wir es wollten.

Bemerkung: Um mehr über Namespace und Scope in Python zu erfahren, schau dir die Dokumentation an.

19.2 Parameter übergeben

Ein wichtiges Konzept, das einen sichtbaren Einfluss auf die Funktionsweise deiner Funktionen haben wird, ist die Übergabe von Parametern. Dies beschreibt die Art und Weise, wie eine Variable verarbeitet wird, wenn man sie an eine Funktion übergibt - in einem *Pass-by-Value* Szenario wird das Argument als neue lokale Variable behandelt und hat keinen Einfluss auf die ursprüngliche Variable (falls eine Variable als Argument übergeben wurde). Im Falle von *Pass-by-Reference* kann die als Argument übergebene Variable innerhalb einer Funktion beeinflusst werden. In Python ist die Methode der Parameterübergabe eine spezielle Kombination aus beidem - Parameter werden per Wert der Objektreferenz übergeben.

Für eine gute Erklärung der Parameterübergabe und den Unterschied zwischen den verschiedenen Techniken, empfehle ich dir diesen Blogpost von Robert Heaton zu lesen.

KAPITEL 20

Klassen und Objekte

Python ist eine objektorientierte Sprache - sie basiert auf dem Konzept der „Objekte“, die verschiedene Felder (Variablen) und Methoden (Funktionen, die diese Variablen verändern) enthalten. Alles in Python ist ein Objekt - egal ob es ein Integer oder ein String ist.

Ein Objekt `object` ist ein praktisches Konzept, um eine Abbildung von etwas Abstraktem zu erzeugen - zum Beispiel ein nützlicher Weg, um Daten geordnet zu speichern.

Erinnert dich das an etwas, das wir schon behandelt haben?

Eine Liste ist ein schönes Beispiel dafür, dass in einem Objekt Daten geordnet gespeichert werden. Wie würdest du ein Listenobjekt beschreiben? Welche Attribute hat es? Wann und wie wird es verwendet?

Tipp: Erinnerst du dich an den Befehl `dir(ClassName)`? Er listet alle Attribute und Methoden einer benötigten Klasse auf, wie zum Beispiel `dir(str)`.

Wenn du feststellst, dass du ein Objekt brauchst, das Python nicht hat, kannst du dein eigenes erstellen. Bevor du ein konkretes Exemplar deines Objekts (eine *Instanz*) verwenden kannst, musst du zuerst eine „Vorlage“ definieren, die beschreibt, wie das Objekt aussehen wird und was es kann. Dieser Prototyp wird als *Klasse* bezeichnet:

```
class Player():
```

Dieses Beispiel zeigt die Vorlage eines Player-Objekts, das im Moment noch leer und nicht sehr nützlich ist. Um es nützlicher zu machen, können wir ihm Attribute hinzufügen - `total_count` ist ein Klassenattribut, das alle Instanzen von Objekten der Klasse Player zählt, indem es jedes Mal, wenn ein neues Player()-Objekt erzeugt wird, einen Wert erhöht.

```
class Player():
    total_count = 0
```

Ein Klassenattribut ist für jede Instanz der Klasse Player gleich, und so kannst du die Gesamtzahl der Spieler über jede erzeugte Instanz herausfinden. Andere Attribute, die nützlich sein könnten, wären Instanzattribute (unterschiedlich für jede Instanz eines Objekts) wie Name und Punktestand. Wie sollen diese definiert werden? Und wie können wir wissen, wann ein neues Objekt erstellt wird, um die `total_count` zu erhöhen?

Es ist möglich, eine `__init__()` Methode für deine Klasse zu definieren, die bei der Erzeugung eines neuen Objekts verwendet wird und die weitere Argumente aufnehmen und den Anfangszustand eines Objekts festlegen kann. Auf diese Weise wird bei der Erzeugung des untenstehenden Objekts `player_1` der anfängliche Punktestand des Spielers Null sein, der Name wird als Argument angegeben und `total_count` wird um eins erhöht.

```
class Player():
    total_count = 0

    def __init__(self, name):
        self.name = name
        self.score = 0
        self.__class__.total_count += 1
```

Außerdem können wir Methoden speziell für unsere Klasse von Objekten definieren. Zum Beispiel die Klassenmethoden `update_score()` und `change_name` um die Werte von `name` und dem Punktestand `score` zu aktualisieren.

```
class Player():
    total_count = 0

    def __init__(self, name):
        self.name = name
        self.score = 0
        self.__class__.total_count += 1

    def update_score(self, score):
        self.score = score

    def change_name(self, name):
        self.name = name
```

Die Erzeugung von Objekten und die Verwendung von Methoden ist ziemlich einfach:

```
# Erzeuge eine Instanz eines Objekts der Klasse Player
player_1 = Player("griever418")
player_2 = Player("r00t")

# Verändere den Punktestand von player_1
player_1.update_score(40)
# Ändere den Namen von player_1
player_2.change_name("bott0m")
```

Nun könntest du dich fragen, warum der Aufruf von Methoden auf `player_1` oder `player_2` nur mit einem Argument funktioniert, während die Methodendefinitionen zwei Argumente haben? Sicherlich gibt Python in diesem Fall einen Fehler aus. Wie du vielleicht schon vermutet hast, wird die Instanz des Objekts - `player_1` - als erstes Argument übergeben, und ist eigentlich gleichbedeutend mit der Anweisung `player.update_score(player_1, 40)`.

Bemerkung: Das Schlüsselwort `self` hat in Python keine besondere Bedeutung, es ist nur eine Art Gewohnheit. Du solltest es benutzen, wenn auch nur aus dem Grund, um deinen Code besser lesbarer zu machen, wenn du nach einiger Zeit wieder darauf zurückkommst (du kannst mehr über die Diskussion von `self` in diesem Blogpost von Guido van Rossum lesen - dem Vater von Python).

Der Zugriff auf Attribute ist für alle Objekte wieder gleich: `obj.attribute_name`. Um zum Beispiel den Namen eines Player-Objekts auszugeben, schreibst du:

```
print(player_1.name)
```

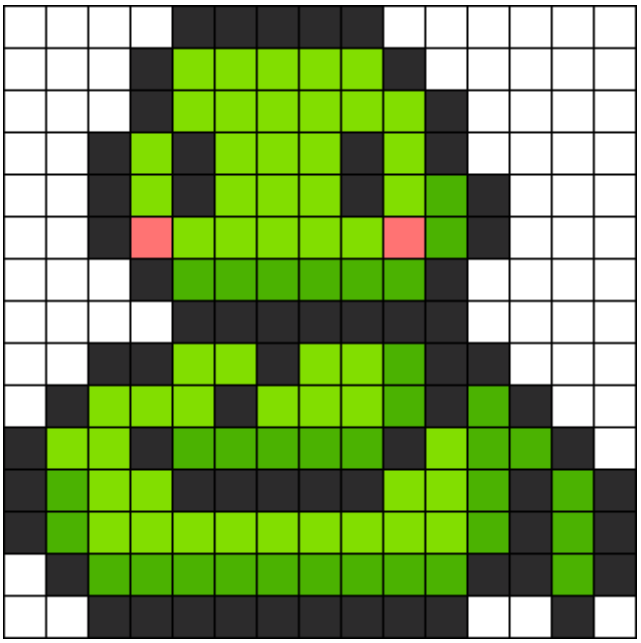
Um ein Attribut für eine Klasse zu erstellen, musst du es nicht in der Klassendefinition deklarieren - sie sind wie lokale Variablen, da sie entstehen, wenn ihnen ein Wert zugewiesen wird. Auf diese Weise können wir ein `counter`-Attribut für unser `player_1` Objekt erstellen. Was gibt das folgende Programm dann aus?

```
player_1.counter = 0

while (player_1.counter < 10):
    player_1.counter += 1

print(player_1.counter)
```

Es gibt noch viele weitere Besonderheiten und nützliche Eigenschaften von Klassen, auf die wir in diesem Tutorial nicht eingehen. Wenn du mehr erfahren willst, schau in die Python Dokumentation.

Um dir ein weiteres Beispiel für die Verwendung von Klassen zu geben, schau dir diese *Snake*-Klasse an, die für eine micro:bit Version des Snake Spiels verwendet werden könnte.

```
class Snake:

        def __init__(self):
            self.x_position = 0
            self.y_position = 0
            self.direction = "w"

        def move_snake(self, x_position, y_position, direction):
            self.x_position = x_position
            self.y_position = y_position
            self.direction = direction

        def show_snake(self):
            display.set_pixel(self.x_position, self.y_position, 9)
            sleep(600)
            display.set_pixel(self.x_position, self.y_position, 0)

# Erstelle eine Instanz eines Snake Objekts unter der Bezeichnung python
python = Snake()

# greife auf seine Position auf der X-Achse zu und gib sie aus
print(python.x_position)

# Bewege python nach rechts
python.move_snake(python.x_position + 1, python.y_position)
```



KAPITEL 21

Schere, Stein, Papier

Wir wollen hier versuchen, das bekannte Spiel *Schere - Stein - Papier* auf unterschiedliche Art und Weise umzusetzen. Dabei lernen wir Schritt für Schritt mittels Python unseren micro:bit zu steuern.

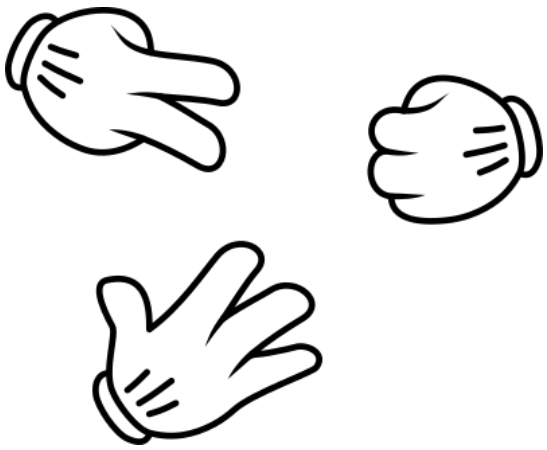


Abb. 1: Source: https://openclipart.org/detail/213382/rockpaperscissors

21.1 SSP 1 - Die Grundlagen

Hier wollen wir den micro:bit nur dazu verwenden, ein entsprechendes Bild am Display anzuzeigen. Das Spiel kann dann mit anderen gespielt werden, nur dass man keine Hände verwendet.

Wir benötigen also 3 Bilder. Diese speichern wir in 3 Variablen. Um anzuzeigen, dass die Werte dieser Variablen im Programm nicht verändert werden sollen, schreiben wir sie in Großbuchstaben. Die Bilder könnten zB so aussehen:

```
from microbit import *

SCHERE = Image("99009:99090:00900:99090:99009:")
STEIN  = Image("09990:99999:99999:09990:00000:")
PAPIER = Image("00000:09990:09990:09990:09990:")
```

Die Auswahl soll mittels der Tasten erfolgen.

- `Taste A` -> zeige die Schere an
- `Taste B` -> zeige den Stein an
- `Tasten A+B` -> zeige das Papier an

Da das Spiel nicht nur einmal zu spielen sein soll, soll der micro:bit andauernd darauf achten, welche Taste gedrückt wurde. Deshalb benötigen wir, wie bei fast allen Programmen die auf einem Mikrochip laufen, eine Endlosschleife mit `while True`.

21.2 SSP 2 - Zufälliges Bild

Dieses mal bestimmen nicht wir, welches Bild angezeigt wird. Der micro:bit soll zufällig eines der 3 Bilder auswählen und zeigen, wenn wir die `Taste A` drücken. So können wir miteinander oder auch selbst gegen den micro:bit spielen.

Spannender wird das, wenn wir auch einen Countdown einbauen, der 3 - 2 - 1 herunterzählt, bevor er das Bild anzeigt! Kannst du das mit einer For Schleife machen?

21.3 SSP 3 - Schütteln und Highscore

Wir können die vorige Variante so abändern, dass alles ganz ohne Taste und nur durch Schütteln des micro:bit funktioniert!

Die 2 Tasten verwenden wir, um den Punktestand mitzuzählen und anzuzeigen.

- `Taste A` zählt die Siege des micro:bits
- `Taste B` zählt deine Siege

21.4 SSP 4 - Der micro:bit lernt die Regeln

Wir kombinieren die ersten 2 Versionen und bringen dem micro:bit die Regeln des Spiels bei.

- Wie in *SSP 1* wählen wir über die Tasten unser Bild, das dann angezeigt wird.
- Jetzt wählt der micro:bit wie in *SSP 2* ein zufälliges Bild, das uns nach einem Countdown angezeigt wird.
- Für uns ist jetzt natürlich klar, wer gewonnen hat. Wir freuen uns, wenn wir gewonnen haben, langweilen uns über ein Unentschieden oder sind traurig, wenn wir verloren haben. Genau diese Reaktionen wollen wir auch von unserem micro:bit sehen!

In einem ersten Schritt müssen wir uns die Regeln als *Wenn . . . , dann . . .* Sätze notieren. Wir stellen uns das Ganze aus der Sicht des micro:bit vor, der glücklich dreinschauen soll, wenn er gewonnen hat. Beim Verlieren soll er natürlich traurig sein und sonst solala. Das könnte, angelehnt an die Schreibweise der Python `if`-Bedingungen, so aussehen:

```
Wenn microbit == spieler:
    Unentschieden ()
Sonst wenn spieler == SCHERE:
    Wenn microbit == STEIN:
        Gewonnen! ()
    Sonst:
        Verloren! ()
Sonst wenn spieler == STEIN:
    ...
```

So kommen wir zu verschachtelten `if`-Bedingungen. Das funktioniert zwar, ist aber nicht sehr elegant. Besser geht das, wenn wir sogenannte Logische Operationen verwenden. Auf diese Weise können wir mit Und `and` bzw. Oder `or` die Regeln schöner formulieren. zB.

```
Wenn (microbit == SCHERE and spieler == PAPIER) or (microbit == STEIN and ...
```

KAPITEL 22

Morsealphabet

22.1 Beschreibung

Das Morsealphabet wurde 1836 von einer Gruppe von Menschen erfunden, zu denen auch der amerikanische Künstler Samuel F. B. Morse gehörte. Mit dem Morsealphabet wird eine Nachricht als eine Reihe von elektrischen Impulsen dargestellt, die entlang von Kabeln zu einem Elektromagneten am empfangenden Ende des Systems gesendet werden können. Die Symbole, die für jeden Buchstaben verwendet werden, sind in der Abbildung unten dargestellt.

A ·–	J ·–––	S ···	1 ·———–
B –···	K –·–	T –	2 ··–––
C –·–·	L ·–··	U ··–	3 ···––
D –··	M ––	V ···–	4 ····–
E ·	N –·	W ·––	5 ·····
F ··–·	O –––	X –··–	6 –····
G ––·	P ·––·	Y –·––	7 ––···
H ····	Q ––·–	Z ––··	8 –––··
I ··	R ·–·	0 –––––	9 ––––·

Abb. 1: Source: raspberrypi.org

Natürlich bist du nicht auf elektrische Impulse beschränkt, du kannst eine Morsebotschaft auch mit Licht oder sogar mit Ton übertragen. Eine Morsenachricht, die über elektrische Kabel gesendet wird, ist als Telegramm bekannt - eine Nachricht, die von einem Funker am sendenden Ende mit einer Morsetaste wie der hier abgebildeten in Morsecode übersetzt wird.

Die Nachricht wird von einem anderen Funker auf der Empfangsseite wieder in normalen Text umgewandelt.

Deine Aufgabe ist es, den micro:bit in eine Maschine zu verwandeln, die Nachrichten mit Morsecode verschlüsseln kann. Wir werden die zu konvertierende Nachricht *plain Text* nennen.

Du wirst das Alphabet mit dem Morsecode in deinem Programm speichern müssen. Du kannst dafür ein Python *Dictionary* verwenden. Hier ist ein Teil eines Python-Dictionarys für Morsecode:

Abb. 2: Morsetaste, Quelle: Wikipedia

```
morse_code = { 'A':'.-',
               'B':'-...',
               'C':'---.',
                ...  }
```

Auf Deutsch bedeutet das: das Zeichen `A` soll durch die Zeichenkette `.-` ersetzt werden; das Zeichen `B` soll durch die Zeichenkette `-...` ersetzt werden und so weiter. Du kannst ein Dictionary so ausgeben:

```
print(morse_code)
```

Probiere das aus, experimentiere mit der REPL.

KAPITEL 23

Cäsar-Verschlüsselung

23.1 Beschreibung

Seit Menschen die Schrift benutzen, versuchen sie, Geheimbotschaften zu verfassen. Eine der frühesten Verschlüsselungen ist als Cäsar-Verschlüsselung bekannt, benannt nach Julius Cäsar, und wurde vom römischen Kaiser benutzt, um mit Truppen auf dem Schlachtfeld zu kommunizieren. Mit der Cäsar-Verschlüsselung verschlüsselst du alle Buchstaben in einer Nachricht, indem du das Alphabet um eine Anzahl von Stellen verschiebst. Die Abbildung unten zeigt, wie man eine Nachricht mit einer Verschiebung von 3 Buchstaben verschlüsselt:

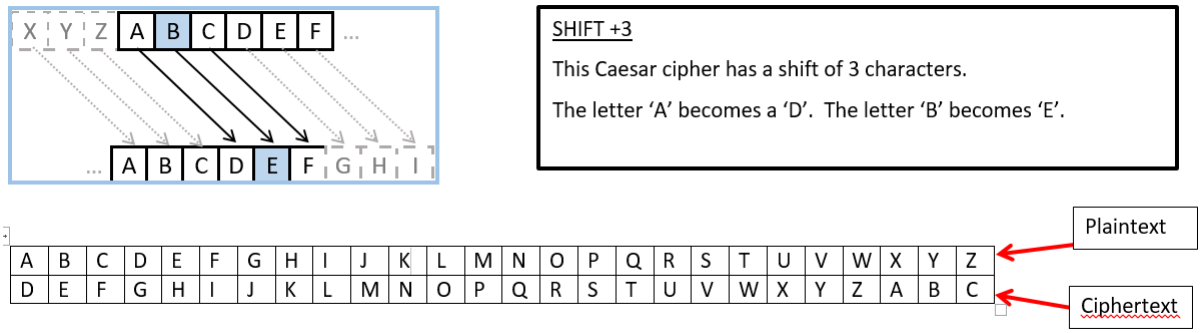


Abb. 1: Source: https://upload.wikimedia.org/wikipedia/commons/f/fa/Ascii-proper-color.svg

Deine Aufgabe ist es, deinen micro:bit in eine Maschine zu verwandeln, die Nachrichten mit der Cäsar-Verschlüsselung **verschlüsseln** kann. Wir nennen die zu verschlüsselnde Nachricht *plain text* und die verschlüsselte Nachricht *cipher text*.

Es gibt einen Trick, den du verwenden kannst, um die Nachricht zu verschlüsseln, oder zu verschieben. Der Trick beruht auf der Tatsache, dass dein micro:bit die Buchstaben des Alphabets als Zahlen sieht. Du kannst einen Buchstaben in eine Zahl übersetzen und wieder zurück, indem du die Python Funktionen `ord()` und `chr()` benutzt.

Sagen wir, du willst jeden Buchstaben um 4 Stellen verschieben. Versuche diesen Code zu benutzen, um das Zeichen in eine Zahl zu verwandeln und 4 zu addieren:

```
ascii_char = ord(plaintext_char) + 4
```

Auf Deutsch heißt das: übersetze `plaintext_char` in eine Zahl mit der Funktion `ord()` und addiere 4, die Anzahl der Zeichen, um die wir verschieben wollen.

Das funktioniert, weil Zeichen in Python als Zahlen kodiert sind. Eines der beliebtesten (und kleinsten) Systeme ist ASCII (American Standard Code for Information Interchange). Wenn du dir jedoch die Tabelle unten ansiehst, kannst du sehen, dass es nur lateinische und einige Sonderzeichen beinhaltet. Deshalb ist das native Kodierungssystem von Python (und den meisten Sprachen) UTF-8, welches auch rückwärtskompatibel zu ASCII ist.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Aber halt, es gibt noch eine Sache, die wir beachten müssen. Da wir nur Großbuchstaben von A-Z akzeptieren, müssen wir sicherstellen, dass wir uns nur innerhalb dieser Grenzen bewegen. Wenn wir 4 zu Z addieren landen wir in der ASCII-Tabelle bei ‚^', deshalb müssen wir von dieser Stelle 26 abziehen um zum *D* zu kommen.

```
ascii_char = ord(plaintext_char) + 4

    if ascii_char > ord('Z')
    ascii_char = ascii_char - 26

    encrypted_char = chr(ascii_char)
```

Versuche nun einige Nachrichten zu verschlüsseln und dann zu entschlüsseln. Du kannst versuchen, den folgenden englischen Text zu entschlüsseln. Kannst du dein Programm dazu bringen, den richtigen Klartext selbständig zu erkennen?

> S kw k csmu wkx… S kw k gsmuon wkx. Kx exkddbkmdsfo wkx. S drsxu wi vsfob rebdc. Rygofob, S nyx'd uxyg k psq klyed wi csmuxocc, kxn kw xyd cebo grkd sd sc drkd rebdc wo. S kw xyd losxq dbokdon kxn xofob rkfo loox, dryeqr S boczomd wonsmsxo kxn nymdybc. Grkd'c wybo, S kw kvcy cezobcsdsyec sx dro ohdbowo; govv, kd vokcd oxyeqr dy boczomd wonsmsxo. (S'w ceppsmsoxdvi onemkdon xyd dy lo cezobcdsdsyec, led S kw.) Xy, csb, S bopeco dy lo dbokdon yed yp gsmuonxocc. Xyg, iye gsvv mobdksxvi xyd lo cy qyyn kc dy exnobcdkxn drsc. Govv, csb, led S exnobcdkxn sd. S gsvv xyd, yp myebco, lo klvo dy ohzvksx dy iye zbomscovi gry sc qysxq dy ceppob sx drsc mkco pbyw wi gsmuonxocc; S uxyg zobpomdvi govv drkd S gsvv sx xy gki „wemu drsxqc ez" pyb dro nymdybc li bocoxdsxq drosb dbokdwoxd; S uxyg loddob drkx kxiyxo drkd li kvv drsc S kw rkbwsxq yxvi wicovp kxn xy yxo ovco. Led cdsvv, sp S nyx'd qoddbokdon, sd sc yed yp gsmuonxocc. Wi vsfob rebdc; govv, drox vod sd rebd ofox gybco!

– P. Nycdyiofcui, Xydoc Pbyw Exnobqbyexn

KAPITEL 24

Substitution

24.1 Beschreibung

Die Verschlüsselung durch Substitutions ist trügerisch einfach. Nachrichten werden mit einem Schlüssel verschlüsselt, der im Voraus erstellt wird. Du erzeugst den Schlüssel, indem du die Positionen der Buchstaben im Alphabet veränderst:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
V	J	Z	B	G	N	F	E	P	L	I	T	M	X	D	W	K	Q	U	C	R	Y	A	H	S	O

Um Nachrichten durch Substitution verschlüsseln und entschlüsseln zu können, musst du einen Schlüssel erstellen, der zur Erzeugung des Chiffriertextes verwendet wird, und ihn speichern. Ein Dictionary könnte eine gute Datenstruktur für diesen Zweck sein.

Ein Python-Dictionary für die obige Substitutions-Verschlüsselung würde etwa so aussehen:

```
key =  {'A':'V',
        'B':'J',
        'C':'Z',
        'D':'B',
          ...   }
```

Versuche deine eigenen Nachrichten zu verschlüsseln und zu entschlüsseln und wenn du dich der Herausforderung gewachsen fühlst, versuche den untenstehenden englischen Text zu entschlüsseln. Um dir die Mühe zu ersparen: der obige Schlüssel gilt nicht für diese Nachricht.

Versuche an Spracheigenschaften oder Methoden zu denken (oder suche danach), die dir helfen könnten, den Schlüssel zu finden.

> O zay vcur xzfyozx fr nb gorfuj rdfr lftm rcttat ydokd youu zcgct ucfgc nc rouu O, raa, fn fr tcjr; “fkkolczrfuub” at ardctyojc. Pctjpflozx rdc yolay rdfr nb kazzchoaz yord dct dpjefzl’j “rckdzokfu nfrrctj” yfj jpvvokoczr ra czroruc nc ra doj nfzpjktowr, O eatc rdc lakpnczr fyfb fzl ecxfz ra tcfl or az rdc Uazlaz eafr. Or yfj f jonwuc, tfneuozx rdozx—f zfogc jfouat’j cvvatr fr f wajr-vfkra loftb—fzl jrtagc ra tckfuu lfb eb lfb rdfr ufjr fyvpu gabfxc. O kfzzar frrcnwr ra rtfzjktoec or gctefron oz fuu orj kuaplozcjj fzl tclpzlfzkc, epr O youu rcuu orj xojr czapxd ra jdcy ydb rdc japzl av rdc yfrct fxfozjr rdc gcjjcu’j jolcj eckfnc ja pzczlptfeuc ra nc rdfr O jrawwcl nb cftj yord karraz. Qadfzjcz, rdfzm xal, lol zar mzay iporc fuu, cgcz rdapxd dc jfy rdc korb fzl rdc Rdozx, epr o jdfuu zcgct juccw kfunub

fxfoz ydcz O rdozm av rdc dattatj rdfr uptm kcfjcucjjub ecdozl uovc oz ronc fzl oz jwfkc, fzl av rdajc pzdfuuaycl eufjwdcnocj vtan culct jrftj ydokd ltcfn eczcfrd rdc jcf, mzayz fzl vfgaptcl eb f zoxdrnftc kpur tcflb fzl cfxct ra uaajc rdcn az rdc yatul ydczcgct fzardct cftrdipfmc jdfuu dcfgc rdcot nazjrtapj jrazc korb fxfoz ra rdc jpz fzl fot.

—D.W. Uagcktfvr, Kfuu Av Krdpudp

KAPITEL 25

Vigenère-Verschlüsselung

Diese Verschlüsselung, auch ‚le chiffre indéchiffrable' genannt, wurde erstmals von Giovan Battista Belazzo beschrieben. Obwohl das Konzept leicht zu verstehen ist, konnte die Verschlüsselung drei Jahrhunderte lang nicht geknackt werden, bis Friedrich Kasiski einen ersten erfolgreichen Generalangriff vorstellte.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Abb. 1: Vigenère Tabelle (Source: https://tinyurl.com/yxmbt48f)

Sie funktioniert ähnlich wie die Cäsar-Verschlüsselung, da sie ebenfalls auf der Verschiebung von Buchstabenpositionen basiert. Anstatt alle Buchstaben um den gleichen Wert zu verschieben, wählen wir diesmal ein Schlüsselwort, das für jeden Buchstaben im Klartext einen anderen Wert bestimmt.

Als Beispiel, nehmen wir an, dass dies unser Klartext ist:

```
ATTACKATDAWN
```

Dann wählst du ein Schlüsselwort (z.B. Snake) und wiederholst es für jede Zeile des Klartextes:

```
SNAKESNAKESN
```

Dann benutzt du jeden Buchstaben des Schlüssels, um die Verschiebung jedes Buchstabens des Klartextes zu bestimmen:

```
Plaintext:  ATTACKATDAWN
Key:        SNAKESNAKESN
Ciphertext: SGTKGCNTNEOA
```

Du kannst versuchen, deine eigenen Nachrichten zu verschlüsseln und zu entschlüsseln. Wenn du Lust auf eine Herausforderung hast, versuche den Schlüssel zu finden und diesen mit der Vigenère-Verschlüsselung verschlüsselten Text zu entschlüsseln (Hinweis: Es ist nicht einfach):

> Wckl'g jwym avr Tlvfqydnxkwn Vyehqgxat oof im lhm nqmna oyrmavz. Vi qtfg gwym hzpdfhs wf p ahschgjxzg idjtawyt jbxivs dhyars zr avr ucktsaiympca dd lbungq tur naqh ucgtq bag vcrhewpprbuu rudxjh bc axyhnxl vhfodl--uhgrs jbms sdpfz.
>
> Hut Fbaquwgdlf'f Vsbks Gd Ral Unayqf oyhm flbgxmgz oyrmavz. Vi qtfg gwym avr qcla rexld pb rmglasarc bz hut ntu unayvawp vyknzr qjtzhrg. Gm zolh rahh gwc xmtrrr hm o cpl zhznrrbj ungeel pypqmlf vh jbrs uptbuu ldsk ifnxll zanhfxk chi zr h gyxax vt ytkhu kepnilr edsgk o yppzl ubab uywpz. Ral uhxbx hzfd rxszf nmn vb jwgvo dyplxag gwc ulgg eyg noypampq tppzss oaylaseh ykl avmcw, ocj bsvo mbj atu skecva hb eyr mce dlx hbq lfta jbasgaoen mknoaxxtawbcq xewfi rh osye whb frwyupzviyml osickdoesq.
>
> Mos Uxrvovvzck'z Uhxbx Ac Gwc Zhznmw llzyh ptavrg zxahrg rahb gwc Xuqlrjhwsqxy Zhznrrbjo.
>
> —Smnnznh Ywhaf, Wgmjvuxixy'g Txswl Hb Ifx Noypvr

KAPITEL 26

Bop-it Spiel

26.1 Beschreibung

In dieser Herausforderung sollst du ein Reaktionsspiel ähnlich wie Bop-it programmieren. Das erste bop-it Spiel, das unten abgebildet ist, wurde in den 1960er Jahren veröffentlicht und seitdem gab es viele Varianten, die deine Reaktionen mit Licht, Sound und Bewegung testen.

Abb. 1: Das originale Bop-it Design, Quelle: Wikipedia

In der Version des Spiels, die du erstellst, muss der Spieler die richtige Taste, `A` oder `B`, in unter 1 Sekunde drücken. Wenn der Spieler die falsche Taste drückt, endet das Spiel. Wenn er die richtige Taste drückt, bekommt er einen Punkt und kann weiterspielen.

KAPITEL 27

Konsonant oder Vokal?

27.1 Beschreibung

In dieser Herausforderung sollst du ein Reaktionsspiel namens *Konsonant oder Vokal* programmieren. Der micro:bit muss dem Spieler einen Buchstaben zeigen, der entweder ein Konsonant oder ein Vokal ist. Der Spieler muss die Taste `A` drücken, wenn es ein Konsonant ist und die Taste `B`, wenn es ein Vokal ist. Man hat nur 1 Sekunde Zeit!

Wenn der Spieler die falsche Taste drückt, zeigt der micro:bit ein passendes Bild oder eine Nachricht an und das Spiel endet.

Wenn sie die richtige Taste drücken, zeigt der micro:bit eine Nachricht oder ein Bild an und der Spieler bekommt einen Punkt und kann weiter spielen.

KAPITEL 28

Fange die Eier

28.1 Beschreibung

In dieser Herausforderung sollst du die Fähigkeit des Spielers testen, ein Ei in einem Korb zu fangen. Stell dir vor es fällt ein Schokoladenei vom Himmel. Der Spieler muss das Ei fangen bevor es auf den Boden fällt. Der Spieler kann sich entweder nach rechts oder links bewegen, indem er die Tasten `A` und `B` auf dem micro:bit benutzt. Wenn das Ei den Boden berührt, endet das Spiel.

KAPITEL 29

Wasserwaage

29.1 Beschreibung

In diesem Projekt werden wir den Beschleunigungssensor nutzen, um eine Wasserwaage herzustellen.

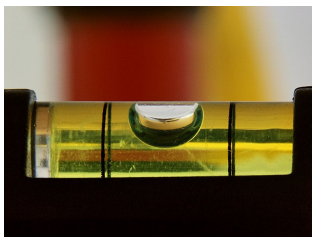


Abb. 1: Quelle: Wikipedia

Eine Wasserwaage, wie die im Bild oben, wird verwendet, um festzustellen, ob eine Oberfläche waagrecht ist. Wenn du eine Wasserwaage auf eine ebene Fläche legst, ruht die Blase in der Mitte der Röhre. Wenn die Oberfläche nach links oder rechts geneigt ist, wird sich die Blase ebenfalls nach links oder rechts bewegen und anschaulich anzeigen, dass es eine Neigung gibt.

Du kannst die vom Beschleunigungsmesser zurückgegebenen Werte dazu benutzen, herauszufinden, ob der micro:bit auf einer ebenen Fläche steht oder nicht. Zeige dem Benutzer die Neigung z.B. mit einem Pfeil an.

KAPITEL 30

Theremin

30.1 Beschreibung

In diesem Projekt werden wir den Beschleunigungsmesser benutzen, um die Frequenz eines Tons zu kontrollieren.

Das Theremin ist ein seltsames und wunderbares elektronisches Instrument, das keinen Körperkontakt benötigt. Das Theremin wurde 1920 von Léon Theremin, einem frühen russischen Elektronik-Ingenieur, erfunden. Es wird gespielt, indem man seine Hände in der Nähe von zwei Antennen bewegt - die erste steuert die Lautstärke des Ausgangs und die zweite die Tonhöhe.

Für die musikinteressierten ist es wissenswert, dass das Theremin Robert Moog zur Erfindung des Synthesizers inspiriert hat. Obwohl es also ein wenig genutztes Instrument ist, hatte es einen starken Einfluss auf die Geschichte der Musik.

Abb. 1: Bild: Leon Theremin, Quelle: Wikipedia

30.1.1 Übrigens: ein kurzer Blick auf Listen

Als Teil dieser Aufgabe musst du einige Werte des Beschleunigungsmessers sammeln und in einer Liste speichern und dann deren Durchschnitt berechnen. Hier sind einige Informationen darüber, wie man das macht.

Eine Liste ist eine Datenstruktur, die in so gut wie allen Programmiersprachen verwendet wird. In unserem Fall, ist es eine nummerierte Sammlung von Beschleunigungssensorwerten. Im Wesentlichen ist es eine Sammlung von Kästchen, in die wir Werte eintragen können - jedes Kästchen hat eine Nummer, beginnend bei 0 und aufsteigend.

Angenommen, wir wollen die letzten 30 Werte des Beschleunigungssensors speichern, dann erstellen wir eine Liste und fügen jedes Mal, wenn wir die Schleife wie folgt durchlaufen, den Wert des Beschleunigungssensors zu dieser Liste hinzu:

```
while True:

    x_acceleration = accelerometer.get_x()

    # Zur Liste der Messwerte hinzufügen
    messungen.append(x_acceleration)

    # Wenn die Messwerte mehr als 30 Werte enthalten, wird der erste Wert vom Anfang␣
↪an gelöscht.
    if len(messungen) > 30:
        messungen.pop(0)
    sleep(1)
```

Wie du sehen kannst, wenn die Liste mehr als 30 Einträge hat, löschen wir nur den ersten Eintrag, Element Nummer 0, mit der Methode `pop()`:

```
messungen.pop(0)
```

KAPITEL 31

Nachricht senden

31.1 Beschreibung

In diesem Projekt werden wir den Funk des micro:bit benutzen, um Nachrichten von einem microbit zum anderen zu senden. Du sollst die Anzahl der Nachrichten, die zwischen einem Paar micro:bits gesendet wurden, auf dem LED Display anzeigen.

KAPITEL 32

Programmierung des micro:bit mit anderen Sprachen

Wie in der Einleitung erwähnt, kann dein micro:bit auch mit JavaScript und C/C++ programmiert werden.

32.1 JavaScript

Online-Editor und Dokumentation für JavaScript findest du auf der Seite von micro:bit.

32.2 C/C++

Der Micro:bit ist mit dem Mbed Online-Compiler programmierbar. Du kannst dir deren Getting Started Video für eine grundlegende Einrichtung ansehen.

KAPITEL 33

Kommandozeile

Genauso wie du eine grafische Benutzeroberfläche (GUI) benutzt, um mit deinem Computer zu interagieren, indem du zB eine Taste am Bildschirm anklickst, ist es möglich, das Gleiche auf einer Kommandozeile zu tun (Command Line Interface - CLI), indem du einen entsprechenden Befehl eingibst.

Die Windows-Eingabeaufforderung **cmd** und die **PowerShell** auf Windws Desktops sind Beispiele für Kommandozeilen. Auch Programmiersprachen-Entwicklungsplattformen wie Python unterstützen eigene Befehlszeilenschnittstellen.

Du wirst beim Coden immer auch mit der Kommandozeile (CLI) arbeiten müssen, daher ist es sinnvoll, dass du dich im Vorfeld mit ihr vertraut machst.

Abb. 1: Die Eingabe von ‚help' in der Windows-Eingabeaufforderung zeigt alle Befehle und was sie tun.

CLIs waren in der Vergangenheit der übliche Weg, um mit Programmen und dem Betriebssystem zu interagieren, da sie viel weniger Systemanforderungen haben. Jetzt benutzt die Mehrheit der Benutzer eine GUI, während CLIs zu einem Werkzeug der fortgeschrittenen Benutzer geworden sind. Obwohl CLIs am Anfang abschreckend und mysteriös aussehen, sind sie ein sehr effizientes Werkzeug für die Durchführung von Konfigurations- und anderen Aufgaben, da ihre Funktionalität über die einer GUI hinausgeht.

Die Art und Weise, wie du die Kommandozeile deines Betriebssystems verwendest, hängt von deinem System ab und wird daher nicht in diesem Tutorial behandelt. Wir ermutigen dich jedoch, die Grundlagen (oder fortgeschrittene Konzepte) auf eigene Faust zu erkunden.

Python-Modulindex

m

Stichwortverzeichnis

M