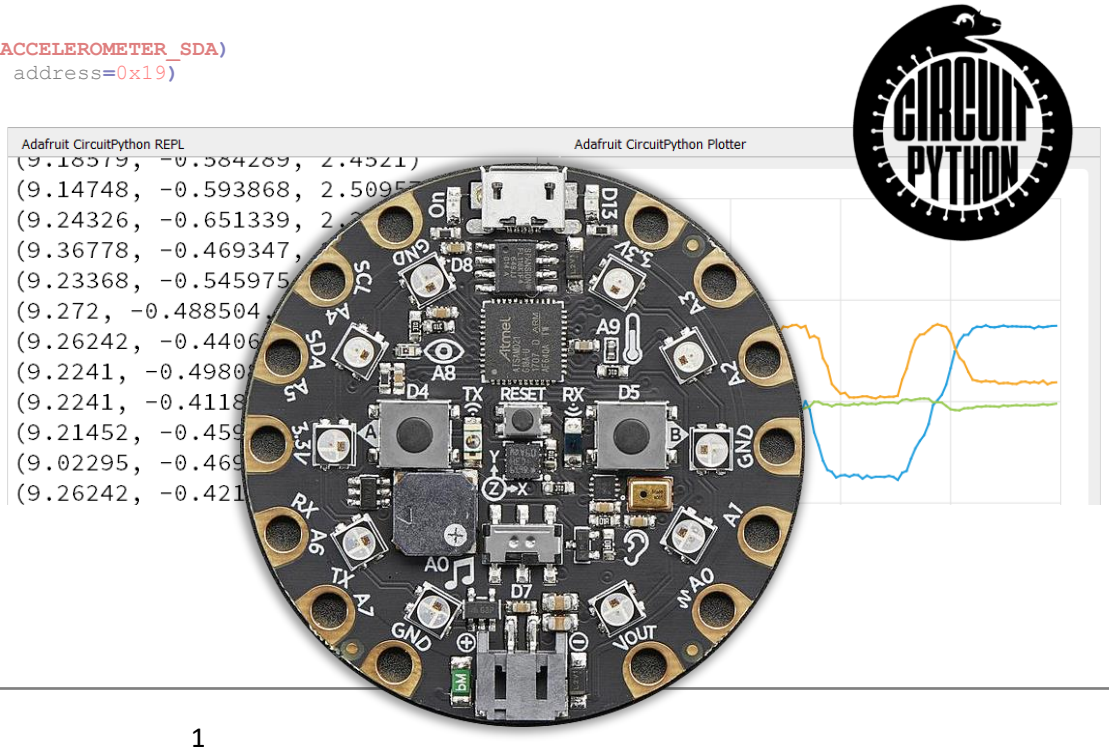
Circuit Python Einführung

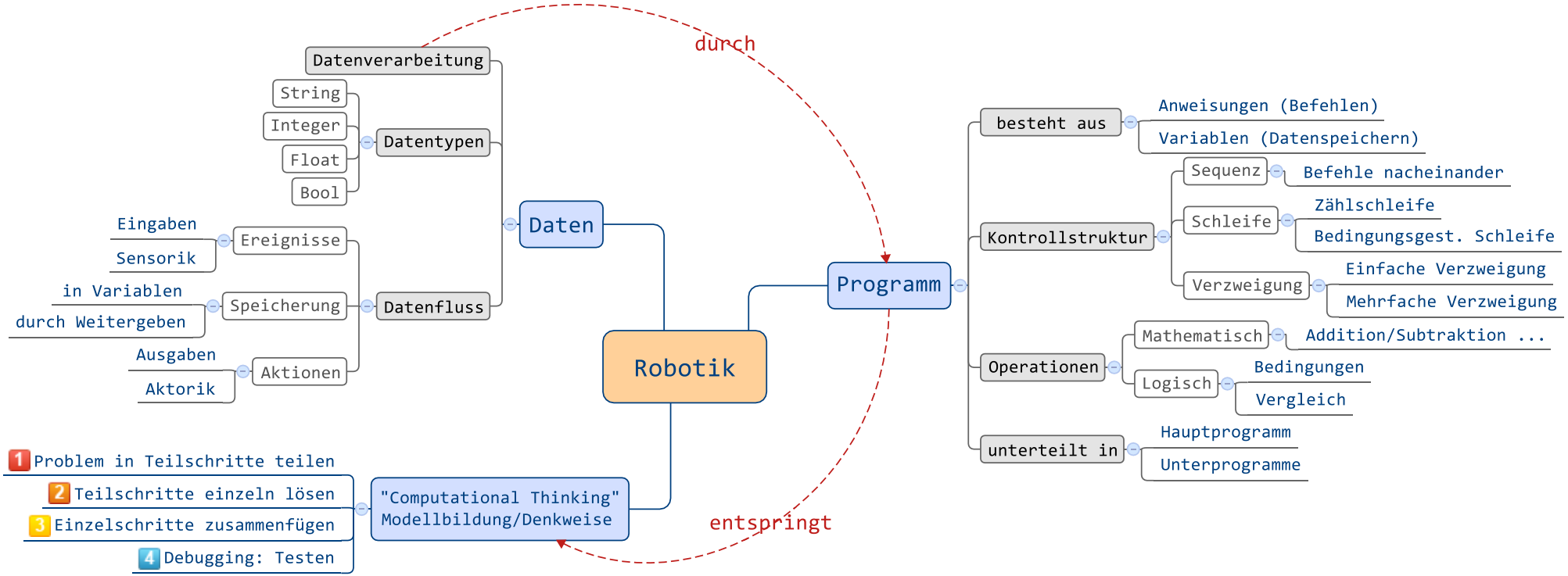
Kepler Gymnasium Weil der Stadt | Thomas Jörg | Stand: 02. September 2018 | Version 1.0

```
import board
import time
import digitalio
import busio
import adafruit_lis3dh
from adafruit_hid.mouse import Mouse

meineMaus = Mouse()

links = digitalio.DigitalInOut(board.BUTTON_A)
links.direction = digitalio.Direction.INPUT
links.pull = digitalio.Pull.DOWN

rechts = digitalio.DigitalInOut(board.BUTTON_B)
rechts.direction = digitalio.Direction.INPUT
rechts.pull = digitalio.Pull.DOWN

i2cbus = busio.I2C(board.ACCELEROMETER_SCL, board.ACCELEROMETER_SDA)
Beschl_Sensor = adafruit_lis3dh.LIS3DH_I2C(i2cbus, address=0x19)

while True:
    if links.value:
        meineMaus.press(Mouse.LEFT_BUTTON)
    elif not links.value:
        meineMaus.release(Mouse.LEFT_BUTTON)

    if rechts.value:
        meineMaus.press(Mouse.RIGHT_BUTTON)
    elif not rechts.value:
        meineMaus.release(Mouse.RIGHT_BUTTON)

    x, y, z = Beschl_Sensor.acceleration
    if x<-1:
        meineMaus.move(x=-int(x*2))
    elif x>1:
        meineMaus.move(x=-int(x*2))

    if y<-1:
        meineMaus.move(y=-int(y*2))
    elif y>1:
        meineMaus.move(y=-int(y*2))
    time.sleep(0.01)
```



Einleitung

Unabhängig von der Programmiersprache schreibt man ein Programm nach allgemeinen Konzepten, die für alle Programmiersprachen gleich sind:



Ein Datenverarbeitendes System (Computer / Roboter / IoT-Device etc.) funktioniert nach dem EVA-Prinzip: Daten-Eingabe, Verarbeitung und Ausgabe. Die Daten-Verarbeitung erfolgt durch Programme. Diese Programme sind nach allgemein gültigen Prinzipien aufgebaut: Man will ein Problem oder eine Aufgabe lösen; die Lösung der Aufgabe wird in Teilschritte aufgeteilt. Diese Teilschritte man einen Algorithmus, und genau diese Teilschritte werden programmiert.

Die verwendete Programmiersprache ist dabei zweitrangig, sie muss nur genügend Anweisungen verstehen und umsetzen können, um das Problem zu lösen!

Datentypen und Variablen: Wie Computer Daten speichern und verarbeiten

Grundsätzlich speichern (und verarbeiten) Computer „unter der Haube“ ihre Daten **immer** in binärer Form, also indem ein Bit (die kleinste Dateneinheit) die zwei Zustände „0“ und „1“ annehmen kann. Daraus werden komplexe Datentypen zusammengesetzt, die unseren menschlichen Gewohnheiten entsprechen.

Damit sich der Computer diese Daten merken kann, gibt man den Daten einen Namen. Das macht man …

- … so wie man zum Beispiel einer Dose eine Beschriftung gibt, damit man weiss, welche Inhalte darin zu finden sind und man sie wiedererkennt.

- … so wie man zum Beispiel in der Mathematik einem Buchstaben in einer Gleichung einen Zahlenwert zuordnet (zum Beispiel `x = 3` oder `x = 4y - 2` oder ähnliches). Die Schreibweise in RobotC ist der mathematischen Schreibweise „abgeschaut“ und deshalb sehr ähnlich.

Wozu benötigt man Variablen?

In Variablen können zum Beispiel Nutzereingaben, Sensorwerte, Uhrzeiten, erreichte Punktzahlen (bei Computerspielen), Kontostände usw. gespeichert werden.

Umgang mit Variablen

Dieses Namen-Wert-Paar nennt man mit dem Fachbegriff eine „**Variable**“. Mit dem **Namen** der Variablen kann das Programm jederzeit auf den **Wert** der Variablen zugreifen. In der Programmierung bedeutet das, dass eine Variable:

- durch die Vergabe eines Namens erzeugt wird *(Deklarierung),*
- durch Ansprechen des Namens einen ersten Wert zugewiesen bekommt *(Initialisierung)*,
- durch Ansprechen des Namens der Wert ausgelesen werden kann *(Lesen)*,
- dieser Wert jederzeit geändert werden kann *(Zuweisung)*.

Prinzip 1: y = 23	Prinzip 2: y = y + 1
Die Variable mit Namen „y“ bekommt den Wert auf der rechten Seite zugewiesen hier die Ganzzahl „23“, ***wie in der Mathematik üblich.***	Die Variable mit Namen „y“ bekommt den Wert der rechten Seite zugewiesen: ihren bisherigen Wert PLUS 1, ***und zwar anders als in der Mathematik üblich***.

Die vier einfachen Datentypen

1: int („Integer-Variable“)

Ganze Zahlen, negativ und positiv

```
Ganzzahl = 15
```

2: float („Float-Variable“)

Fließkommazahlen, negativ und positiv

```
Kommazahl = 3.1415926
```

3: string („String-Variable“)

Buchstaben und Text

```
meinTxt = "Hi \n Welt"
```

4: boolean („Boolsche Variable“)

Zwei logische Werte „True“ & „False“

```
JaOderNein = True
```

Bei Python gilt: Einer Variablen kann jederzeit ein anderer Datentyp zugeordnet werden. Die Sequenz z.B. ergibt keinen Fehler:

```
meineVariable = 15
meineVariable = „ein neuer Text“
```

Diese Programmierweise sollte man aber vermeiden, damit es nicht zu unbeabsichtigten Fehlern kommt, während das Programm läuft.

Ein guter und robuster Programmierstil ist es daher, den Datengtyp einer Variable niemals zu ändern!

Wichtig zu wissen: Verarbeitung im Prozessor

Die CPU (also der Prozessor, das „Gehirn“ des Computers) besitzt eigene, baulich voneinander abgetrennte Teile zur Verarbeitung von Ganz- und Kommazahlen. Ganzzahlen werden schnell und genau, Kommazahlen eher langsam und unpräzise verarbeitet.

Aufgabe zu Datentypen und Variablen

Variablennamen können in einer Programmiersprache frei gewählt werden. Am besten so, dass der Name einen eindeutigen Hinweis auf die Art der Daten gibt. Fülle die Tabelle aus, welcher Datentyp sich hinter den folgenden Variablennamen verbergen sollte/könnte und gib ein Beispiel:

`PaketAnzahl = _____`	`istHeuteOstern = _____`
`SchuelerAnzahl9a = _____`	`Austragungsort = _____`
`MeinGewicht_kg = _____`	`istLuege = _____`
`Automarke = _____`	`Guthaben = _____`

Anweisungen

Ein Programm besteht aus einzelnen Anweisungen. Jede Anweisung wird vom Computer ausgeführt. Beispiele für Anweisungen in Python sind:

Anweisung	Was der Computer damit macht
`print(" Hallo \n Welt")`	Ausgabe von `Hallo` `Welt` am Bildschirm
`pi = 3.1415` `umfang = 2 * pi`	Zuweisung des Float-Zahlenwerts 3.1415 der Variable pi, anschließend Zuweisung des Float-Zahlenwerts 6,2830 der Variable umfang
`import board`	Importieren einer Bibliothek in das aktuelle Programm

Regel in Python:

Eine Anweisung wird in eine Zeile geschrieben. Der Zeilenumbruch zeigt dem Python-Interpreter an, dass die Anweisung beendet ist und nun eine neue Anweisung folgt.

Objekte und Objektschreibweise in Python

Ein zentrales Konzept vieler moderner Programmiersprachen ist die sogenannte Objektorientierung. Zum Beispiel Java, Python, C++, Javascript usw. nutzen diese.

Grundlegender Gedanke der sogenannten Objektorientierung ist:

Objekte sind Dinge, welche Eigenschaften haben (das sind objekteigene Variablen), etwas tun können (das nennt man eine Methode eines Objekts) und eventuell weitere Objekte haben. Diese Objekte kann man in einer Programmiersprache modellieren, also vereinfacht nachbauen. So bildet man ein reales Objekt in einer Programmiersprache ab.

Ein Auto in der realen Welt zum Beispiel

- … hat die Eigenschaften Lackfarbe, Höchstgeschwindigkeit, Momentangeschwindigkeit
- … kann Beschleunigen, Bremsen, Abbiegen
- … besitzt ein Kennzeichen, einen Motor und ein Lenkrad – also wiederum Objekte, welche Eigenschaften und Methoden besitzen.

In einer normalen umgangs-sprachlichen Beschreibung würde man sagen:	In Python würde man sagen:
Die Lackfarbe des Autos ist grün.	`Auto.Lackfarbe = „gruen“`
Die Höchstgeschwindigkeit des Autos beträgt 200 km/h.	`Auto.Hoechstgeschwindigkeit = 200`
Die Momentangeschwindigkeit des Autos liegt bei 100 km/h.	`Auto.Momentangeschwindigkeit = 100`
Das Auto soll jetzt beschleunigen.	`Auto.beschleunigen()`
Das Auto soll jetzt bremsen.	`Auto.bremsen()`
Das Auto soll jetzt abbiegen.	`Auto.abbiegen()`
Das Auto hat ein Kennzeichen mit der Aufschrift „S-XY-22“.	`Auto.Kennzeichen.Aufschrift = „S-XY-22“`

Auf Bestandteile eines Objektes – seien es nun Eigenschaften, Methoden oder weitere Objekte – wird mit einem sogenannten **Punktoperator** zugegriffen. Man spricht bei der Nutzung des Punktoperators von der sogenannten „**Dot**“-Syntax („**Dot**“ ist das englische Wort für „Punkt“).

Aufgabe zum Punktoperator

Fülle die fehlenden Tabellenzellen aus:

Das Handy ist angeschaltet.	
	`Neopixel.brightness = 0.5`
Der Schalter des Elektroherds steht auf der Stellung „zwei“	
	`Katze.Schnurren()`
Die Katze besitzt gelbe Augen.	
	`Kantine.Mittagessen.Beilage = „Pommes“`

Kontrollstruktur Teil 1: Sequenz von Anweisungen

Ein Computerprogramm besteht aus einer Abfolge von Anweisungen, das nennt man Sequenz. Diese Anweisungen werden der Reihe nach, von oben nach unten abgearbeitet. Ab und zu kommen Sprünge vor, in denen das Programm an eine andere Stelle springt. Dort wird wieder sequentiell weiter abgearbeitet.

```
import board                                    # Objekt board: Befehle für den Board-Zugriff laden
import digitalio                                # Objekt „Pin-Daten“ des Boards dazuladen
import time                                     # Objekt time: Zeit-Funktionen dazuladen

meinPin = digitalio.DigitalInOut(board.D13)     # der neuen Variable led aus den Pin-Daten des Boards den Pin board.D13
                                                # zuweisen. Nun hat die Variable led den Zugang zum Pin gespeichert

meinPin.direction = digitalio.Direction.OUTPUT # Der Pin wird als Ausgang geschaltet.

meinPin.value = True                            # Der Pin wird angeschaltet.
time.sleep(1)                                   # Das Objekt time hat eine Methode „sleep“: Es wird eine Sekunde gewartet.
```

Kontrollstruktur II: Schleifen

Will man Sequenzen häufiger wiederholen, nutzt man Schleifen. Beispiel: eine LED unendlich oft blinken lassen, dazu schaltet man sie immer wieder an und aus.

Zählschleife:

Eine Variable wird bei jedem Schleifendurchlauf hochgezählt bis zum Maximalwert. Ist dieser erreicht, wird die Schleife beendet.

```
import time
from adafruit_circuitplayground.express import cpx


for i in range (1, 5, 1):
    cpx.start_tone(262)
    time.sleep(0.2)
    cpx.stop_tone()
    time.sleep(0.2)
    print(i)
```

Was das Programm macht:

Man hört 4-mal (nicht 5-mal!) einen Ton und es werden die Zahlen `1, 2, 3, 4` nacheinander ausgegeben.

Indents in Python

Damit der Python-Interpreter weiss, welche Anweisungen zur Schleife gehören, müssen diese eingerückt sein. Dieses Prinzip ist innerhalb von Python allgemeingültig, es gilt für alle Funktionen, Schleifen, Verzweigungen, Klassen usw.

Bedingungsgesteuerte Schleife

Ob die Schleife wiederholt wird, muss mit einer Bedingung geprüft werden.
Ist die Bedingung wahr, wird die Schleife wiederholt; ist die Bedingung falsch, wird die Schleife beendet.

```
import time
from adafruit_circuitplayground.express import cpx

i = 1
while i<5:
    cpx.start_tone(262)
    time.sleep(0.2)
    cpx.stop_tone()
    time.sleep(0.2)
    print(i)
    i = i+1
```

Was das Programm macht:

Man hört 4-mal (nicht 5-mal!) einen Ton und es werden die Zahlen `1, 2, 3, 4` nacheinander ausgegeben.

Eine Falle bei while-Schleifen

Vergisst man hier die letzte Zeile `i = i+1`, so wird die Variable i immer auf dem initialisierten Wert 1 bleiben und die while-Schleife wird niemals unterbrochen. Das Programm wird sich also in dieser Schleife „aufhängen“.

Kontrollstruktur Teil III: Verzweigungen

Wenn zum Beispiel der linke Knopf gedrückt wird, soll der Computer Piepsen und wenn der rechte Knopf gedrückt wird, eine LED blinken lassen. Zum jeweiligen Knopfdruck macht der Computer also etwas anderes, es werden also unterschiedliche Sequenzen abgearbeitet. Dazu benötigt man Verzweigungen.

Die einfache Verzweigung

Eine Bedingung wird geprüft. Ist sie wahr, wird Sequenz 1 abgearbeitet, ist sie falsch, wird Sequenz 2 abgearbeitet.

```
import board
import time
import touchio
from adafruit_circuitplayground.express import cpx

B_A1 = touchio.TouchIn(board.A1)
B_A2 = touchio.TouchIn(board.A2)

while True:
    if B_A1.value == True:
        cpx.start_tone(262)
    else:
        cpx.stop_tone()

    if B_A2.value == True:
        cpx.red_led = True
    else:
        cpx.red_led = False

    print(B_A1.value, B_A2.value)
    time.sleep(0.1)
```

Was das Programm macht:

Am CPX werden die beiden Touchsensoren an Pin A1 und Pin A2 angeschaltet. Die while-Schleife läuft unendlich oft. Innerhalb der while-Schleife wird abgefragt, ob A1 oder A2 berührt sind. Ist A1 berührt, so wird ein Ton gestartet. Ist A1 nicht berührt, wird der Ton gestoppt. Ist A2 berührt, so leuchtet die rote LED. Ist A2 nicht berührt, wird die LED ausgeschaltet. Zuletzt wird der aktuelle Berührungszustand von A1 und A2 ausgegeben.

Die Mehrfachverzweigung

Eine Bedingung wird auf viele unterschiedliche Ergebnisse hin geprüft und dann jeweils eine zugehörige Sequenz abgearbeitet.

```
import board
import time
import touchio
from adafruit_circuitplayground.express import cpx

B_A1 = touchio.TouchIn(board.A1)
B_A2 = touchio.TouchIn(board.A2)
B_A3 = touchio.TouchIn(board.A3)
B_A4 = touchio.TouchIn(board.A4)
B_A5 = touchio.TouchIn(board.A5)
B_A6 = touchio.TouchIn(board.A6)

while True:
    if B_A1.value == True:
        cpx.start_tone(262)
    elif B_A2.value == True:
        cpx.start_tone(294)
    elif B_A3.value == True:
        cpx.start_tone(330)
    elif B_A4.value == True:
        cpx.start_tone(349)
    elif B_A5.value == True:
        cpx.start_tone(392)
    elif B_A6.value == True:
        cpx.start_tone(440)
    else:
        cpx.stop_tone()
time.sleep(0.1)
```

Kontrollstruktur Teil IV: Funktionen, die Grundlagen

```
import board
import time
import touchio
from adafruit_circuitplayground.express import cpx

B_A1 = touchio.TouchIn(board.A1)
B_A2 = touchio.TouchIn(board.A2)
B_A3 = touchio.TouchIn(board.A3)
B_A4 = touchio.TouchIn(board.A4)

def allemeine():
    cpx.play_tone(262,0.2)
    cpx.play_tone(294,0.2)
    cpx.play_tone(330,0.2)
    cpx.play_tone(349,0.2)

def entchen():
    cpx.play_tone(392,0.4)
    cpx.play_tone(392,0.4)

def schwimmenAufDemSee():
    cpx.play_tone(440,0.2)
    cpx.play_tone(440,0.2)
    cpx.play_tone(440,0.2)
    cpx.play_tone(440,0.2)
    cpx.play_tone(392,0.4)

while True:
    if B_A1.value == True:
        allemeine()
    if B_A2.value == True:
        entchen()
    if B_A3.value == True:
        schwimmenAufDemSee()
    if B_A4.value == True:
        allemeine()
        entchen()
        schwimmenAufDemSee()
    time.sleep(0.1)
```

Um häufig wiederkehrende Aufgaben zusammenzufassen, nutzt man Funktionen. Eine Funktion ist eine Anweisungssequenz, die unter einem eigenen Namen abgespeichert wird. Schreibt man nun im normalen Programmablauf den Namen der Funktion, so wird die gesamte darunter abgespeicherte Befehlssequenz abgespielt. Nutzt man Funktionen, erspart dies beim Programmieren viel Zeit und vereinfacht ein Programm erheblich. Funktionen können nämlich immer wieder aufgerufen und wiederverwendet werden.

Was das Programm macht:

Am CPX werden die vier Touchsensoren A1, A2, A3 und A4 angeschaltet.

Es werden drei Funktionen definiert, nämlich

`allemeine(), entchen(), schwimmenAufDemSee()`

Wenn man eine dieser Funktionen aufruft, so spielen sie jeweils eine einprogrammierte Tonsequenz ab.

In der Endlos-while-Schleife werden die Berührungen von A1 bis A4 abgefragt. Beim jeweiliger Berührung wird eine der obigen Funktionen aufgerufen. Lediglich bei A4 werden nacheinander alle Funktionen abgespielt.

Funktionsdefiniton

Man platziert das Schlüsselwort def, dann den Variablennamen, dann ein Klammerpaar und zuletzt einen Doppelpunkt. Nun hat man den Funktionskopf definiert. Im sogenannten Funktionsrumpf wird nun die Anweisungssequenz einprogrammiert. Ruft man im späteren Programm den Rumpf (also den Namen) auf, so wird die jeweilig einprogrammierte Anweisungssequenz abgespielt.

```
import board, time, touchio
from adafruit_circuitplayground.express import cpx

B_A1 = touchio.TouchIn(board.A1)
B_A2 = touchio.TouchIn(board.A2)
B_A3 = touchio.TouchIn(board.A3)

def allemeineEntchen(tempo):
    cpx.play_tone(262,tempo)
    cpx.play_tone(294,tempo)
    cpx.play_tone(330,tempo)
    cpx.play_tone(349,tempo)
    cpx.play_tone(392,tempo*2)
    cpx.play_tone(392,tempo*2)

while True:
    if B_A1.value == True:
        allemeineEntchen(0.1)
    if B_A2.value == True:
        allemeineEntchen(0.2)
    if B_A3.value == True:
        allemeineEntchen(0.4)
    time.sleep(0.1)
```

Kontrollstruktur Teil V: Funktionen mit Parametern

Einer Funktion kann man Parameter übergeben, die von der Funktion als Variable entgegengenommen werden. Entweder einen oder mehrere Parameter sind möglich.

Was das Programm macht:

Am CPX werden die drei Touchsensoren A1, A2 und A3 angeschaltet.
Von jeder Berührung wird dieselbe Funktion `allemeineEntchen(tempo)` aufgerufen, jeweils aber mit einem anderen Parameter, den die Funktion unter dem Variablen-Namen `tempo` entgegennimmt. Mit diesem Tempo wird jeweils die Abspielgeschwindigkeit der Tonsequenz bestimmt.

Kontrollstruktur Teil VI: Funktionen mit Rückgabeparameter

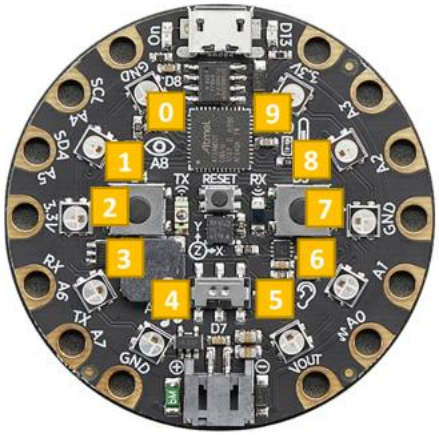
Einer Funktion kann nicht nur ein Parameter übergeben werden, eine Funktion kann auch selbst Werte zurückgeben. Diese Rückgabe erfolgt an eine Variable.

Was das Programm macht:

Am CPX werden die drei Touchsensoren A1, A2 und A3 angeschaltet.
Von jeder Berührung wird dieselbe Funktion `berechneQuadrat(x)` aufgerufen, jeweils aber mit einem anderen Parameter, den die Funktion unter dem Variablen-Namen `x` entgegen nimmt. Von diesem Zahlenwert wird das Quadrat berechnet und im Hauptprogramm an die Variable y zurückgegeben. Dann wird im Hauptprogramm der jeweilige y-Wert ausgegeben.

```
import board, time, touchio
from adafruit_circuitplayground.express import cpx

B_A1 = touchio.TouchIn(board.A1)
B_A2 = touchio.TouchIn(board.A2)
B_A3 = touchio.TouchIn(board.A3)

def berechneQuadrat(x):
    return x*x

while True:
    if B_A1.value == True:
        y = berechneQuadrat(2)
        print("das Quadrat von 2 ist: ", y)
    if B_A2.value == True:
        y = berechneQuadrat(3)
        print("das Quadrat von 3 ist: ", y)
    if B_A3.value == True:
        y = berechneQuadrat(4)
        print("das Quadrat von 4 ist: ", y)
    time.sleep(0.1)
```

Variablen Teil II: Listen, Tupel und Arrays



Der CPX besitzt 10 Neopixel; die Helligkeiten dieser Neopixel werden in Zeile 2 in eine Liste geschrieben. Will man nun den 4. Neopixel ansprechen, so sucht man sich den entsprechenden Eintrag aus der Liste heraus und setzt so dieHelligkeit.

```
import board, time, neopixel
NeopixelListe = neopixel.NeoPixel(board.NEOPIXEL, 10)

for i in range (10):
    NeopixelListe[i] = (0,64,0)
    print(NeopixelListe[i])
    time.sleep(0.1)
print(NeopixelListe)
```

In der 2. Zeile: `NeopixelListe = neopixel.NeoPixel(board.NEOPIXEL, 10)` wird die Liste der Neopixel definiert; hier sind alle Helligkeitswerte abgespeichert.

Die Zeile `print(NeopixelListe)` erzeugt folgende Ausgabe:

```
[(0, 64, 0), (0, 64, 0), (0, 64, 0), (0, 64, 0), (0, 64, 0), (0, 64, 0), (0, 64, 0), (0, 64, 0), (0, 64, 0), (0, 64, 0)]
```

Helligkeit Neopixel 0 | Helligkeit Neopixel 1 | Helligkeit Neopixel 2 | Helligkeit Neopixel 3 | Helligkeit Neopixel 4 | Helligkeit Neopixel 5 | Helligkeit Neopixel 6 | Helligkeit Neopixel 7 | Helligkeit Neopixel 8 | Helligkeit Neopixel 9

Die Zeile `print(NeopixelListe[4])` erzeugt folgende Ausgabe:

```
(0, 64, 0)
```

Die Helligkeit von Neopixel Nummer 4 ist also ein Teil, bzw. einer von 10 Einträgen der gesamten Liste.

Ganz allgemein sind Listen eine Sammlung von Werten, die unter einem einzigen Variablennamen abgespeichert werden und über die Nummer des jeweiligen Listeneintrages angesprochen werden können. Nebenstehendes Programm liest die Zahlenwerte für die Tonhöhe und Tonlänge aus zwei Listen und spielt danach die Töne ab.

```
import board, time
from adafruit_circuitplayground.express import cpx

tonhoehenListe = [262, 294, 330, 349, 392, 392]
tonlaengenListe = [0.1, 0.1, 0.1, 0.1, 0.3, 0.3]

for i in range (len(tonhoehenListe)):
    cpx.play_tone(tonhoehenListe[i],
tonlaengenListe[i])
    time.sleep(0.1)
```

Kurzprogramm, nützlich zu wissen: *Wie schnell wird eigentlich eine Schleife wiederholt?*

```
import board, time

startzeit = time.monotonic()
zeitunterschied = 0
anzahlDurchlaeufe = 0

while zeitunterschied<=1.0:
    zeitunterschied = time.monotonic() - startzeit
    anzahlDurchlaeufe = anzahlDurchlaeufe + 1

print("Anzahl Schleifendurchlaeufe pro Sek.:", anzahlDurchlaeufe)
```

Was das Programm macht:

Mit der Funktion time.monotonic() erhält man auf eine hunderdstel Sekunde genau die Zeit, die der CPX bereits angeschaltet ist. Will man die Zeitspanne einer Sekunde bestimmen, so schaut man nach dem Zeit-Unterschied von zwei Zahlenmesswerten, indem man den aktuellen Zeitwert vom Startzeitwert abzieht.

Lässt man nun eine while-Schleife sooft sich wiederholen, bis eine Sekunde vorbei ist, so kann man eine Größenordnung dafür finden, wie viele Befehle der CPX pro Sekunde in etwa abarbeiten kann.

In der Ausgabe ist zu lesen:

```
Anzahl an Schleifendurchlaeufen in einer Sekunde : 4675
```

Interpretation des Zahlenwertes

Das bedeutet, dass:

- in einer Sekunde diese drei Code-Zeilen insgesamt 4675 mal durchlaufen werden
- Drei Codezeilen benötigen also 1/4675 Sekunden ≅ 0,000213 Sekunden
- Eine Codezeile benötigt ein Drittel davon, also 0,000213/3 Sekunden ≅ 0,00007 Sekunden ≅ 0,07 Millisekunden ≅ 70 Millionstel Sekunden

Inhaltsverzeichnis