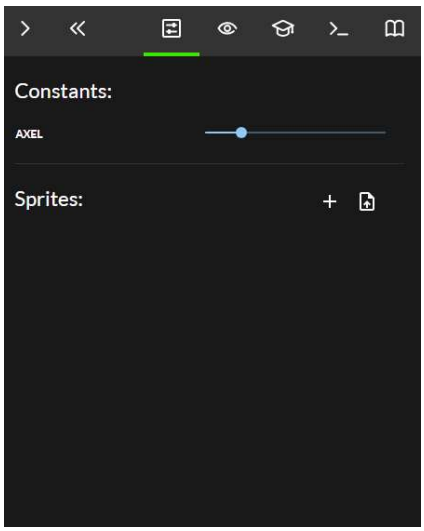
NanoPy Kurzreferenz

Aufruf der Programmierumgebung mit https://editor.nanopy.io. (Funktioniert nicht mit Firefox)

Rechte Seite 

Konstanten



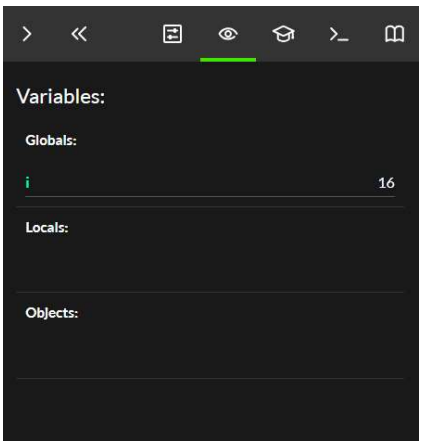
Wird eine Konstante mit einem Wertebereich nach dem Kommentarzeichen festgelegt,

```
const AXEL = 2 # 0 .. 10
```

erscheint diese im Fenster und kann mit dem Slider verändert werden

```
drawText(0,0,AXEL)
update()
```

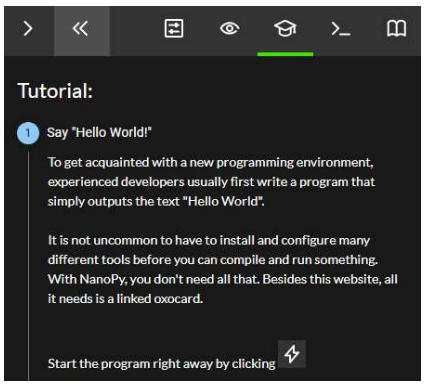
Variablen



Zeigt die Variablen während des Debuggings an

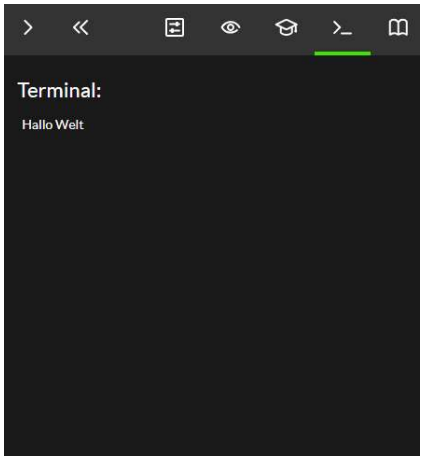
```
1   i = 12
2   while true:
3       print(i)
4       i = i +1
5
```

Tutorial



Zusätzliche Erklärungen eines Programmes. Wird vom Anbieter des Programms erstellt.

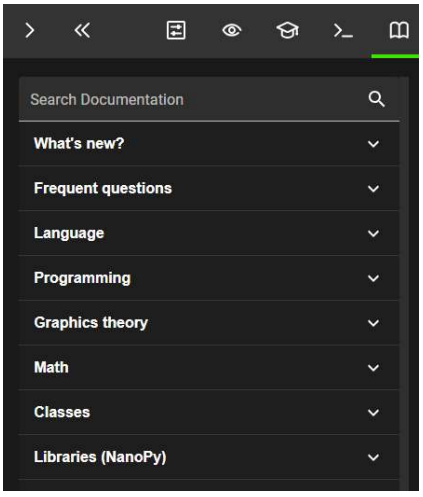
Terminal



Ausgabe von Informationen, die im Programm mit print definiert wurden

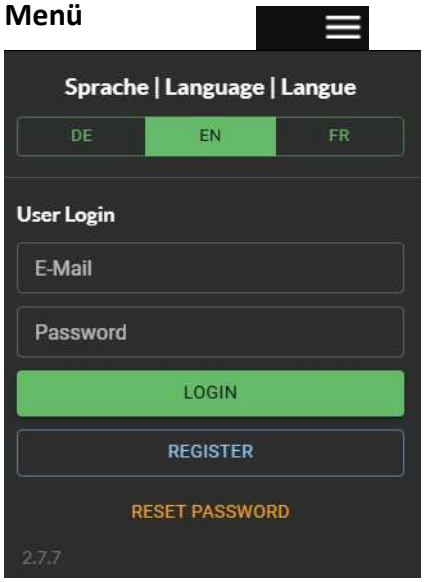
```
print("Hallo Welt")
```

Dokumentation



Hier ist die komplette Dokumentation abgelegt

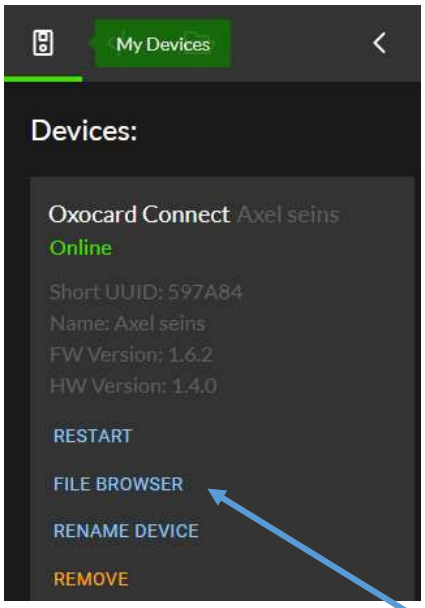
Menü



Auswahl der Sprache
Benutzer Login
Passwort zurücksetzen

Linke Seite 

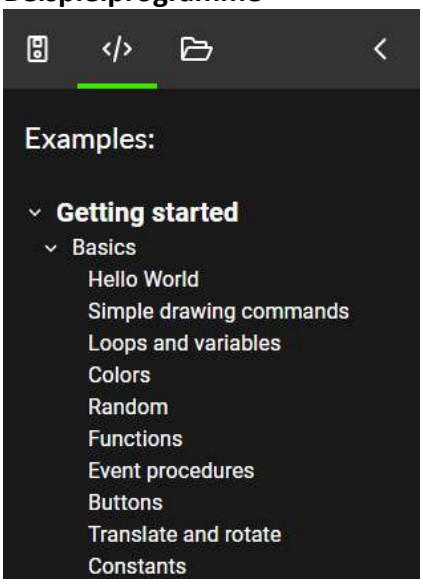
Meine Geräte

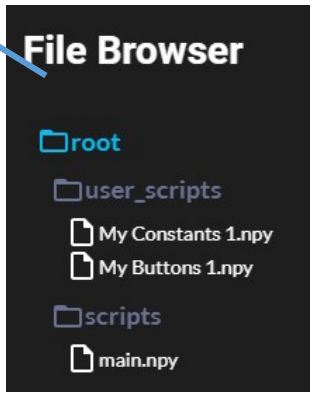


Anzeige der Karte
Status der Karte (Online)

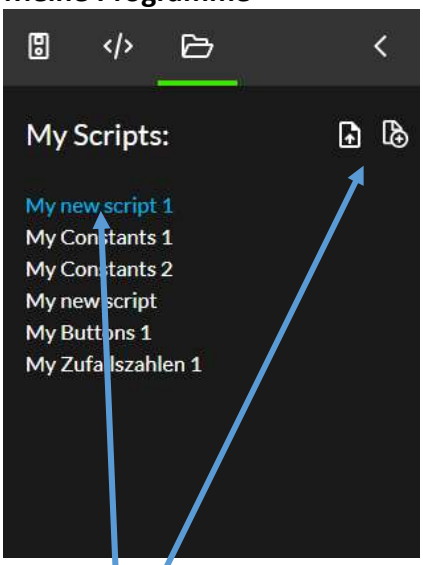
- Neustart der Karte
- Dateisystem der Karte
- Umbenennen der Karte
- Entfernen der karte

Beispielprogramme





Meine Programme

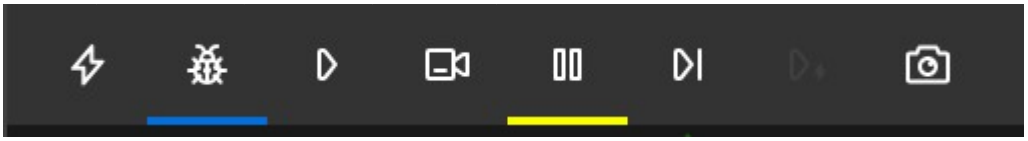


Laden vom PC
Neues Script

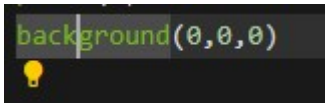
- Laden auf Zusatzkarte
- Laden auf Oxocard
- Sichern auf PC
- Umbenennen
- Löschen

Mitte 

Programmstart
Initialisierung Debugger
Starten Debugger
Observationsmodus
Pause
Schritt vorwärts
Bis zum Breakpoint
Bildschirmfoto erstellen



Hilfe aufrufen:
Glühlampe anklicken

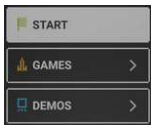


Ein Schritt Rückgängig: Strg + Z

Systemfunktionen		

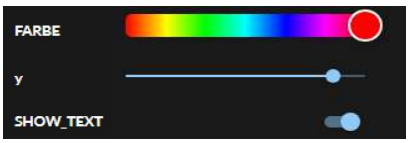
Text auf Konsole ausgeben	print(„Hallo Welt!“) #Text
	print(123) #Zahl
	a = 1.23
	print(a) #Variable
		Sendet den definierten String-Parameter ans Terminal

Ausgabe IP-Adresse	print(ipAddress()) # „192.168.178.128“	Gibt die aktuelle IP-Adresse als String zurück
Ausgabe Hardware	print(getHardwareType()) # 4	gibt den Typ der Oxocard zurück; Galaxy=0,Artwork=1,Science=2, Sience_Plus=3,Connect=4
Ausgabe Sprache	print(getLanguage()) # 1	Gibt die aktuell Systemsprache zurück C_LANGUAGE_EN; DE; FR
Ausgabe Verbindung Internet	print(isConnected()) -> bool	Gibt true zurück wenn die Karte mit dem Internet verbunden ist
Oxocard ausschalten	turnOff()	Schaltet die Oxocard aus
Autostartfunktion	setAutostart()	Beim Neustart wird direct ins Startmenü gesprungen
Pause	print(“Hallo”) delay(1000) print(“Welt”)	Zeitangabe im ms

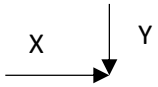
Programm verlassen
 
< WEITER EXIT>
	def onDraw():
	delay(10)
	if getButton():
	if returnToMenu():
	return
		Ereignisfunktion als Dauerschleife
		Pause
		Abfrage ob Button gedrückt
		Abfrage ob weiter oder Exit
		Wenn Exit Hauptmenü

Variablen		
Variable	A = true B = 10 C = 3.14 D = 45.678 E = 3597.34987	Datentyp bool false/true 0 od. 1 Datentyp byte 0 bis 255 Datentyp int -32768 bis 32767 Datentyp long -2147483648 bis 2147483647 Datentyp float 1.17549e-038 bis 3.40282e+038

Auromatischer Wechsel zwischen true und false	blink = false
	initGPIO(C_PIN_05, OUTPUT)
	def onDraw():
	blink = not blink
	if b link:
	writeGPIO(C_PIN_05, 1)
	else:
	writeGPIO(C_PIN_05, 0)
	delay(500)
		erste Festlegung von blink (false)
		Initialisierung Port5 für LED
		Ereignisfunktion als Dauerschleife
		Umkehrung von blink (true/false)
		Abfrage ob blink true
		LED an
		Abfrage ob blink false
		LED aus
		Pause o,5 Sek.

Konstanten	const FARBE = 254 # HUE
	const y = 88 # 10..100
	const SHOW_TEXT = true # true, false
	stroke(FARBE, 255,255)
	if SHOW_TEXT:
	drawText(10,y,"HALLO")
	update()
		
		  Werte lassen sich unter Variablen ändern, wenn hinter # der Werte-bereich angegeben wird

Listen	liste = [1, 2, 3, 4] print(liste[2]) #3 print sizeof(liste) #4 liste[2] = 9	Erzeugung der Liste Abruf des Elementes 2 Ausgabe der Länge der Liste Einen Listenwert ändern 3 -> 9

Textausgabe auf Display		
Textausgabe auf Bildschirm 	drawText(10, 10, "Testtext") update()	Textposition x,y und Text mit update() wird die Textausgabe ausgeführt
Textgröße	textFont(FONT_ROBOTO_BOLD_48) drawText(10, 10, "Testtext") update() # FONT_ROBOTO_16/_24/_32/_48/_80	bestimmt die Schriftgröße (16 bis 80) (BOLD für Fett) FONT_ROBOTO_BOLD_16/_24/_32 FONT_ROBOTO_BOLD_48/_80
Steuerzeichen	drawText("Axel\nChobe") update()	\n = Zeilenwechsel; \# = Raute \t = Tabulator; \‘ = Hochkommata
Textausrichtung	textAlignment(TEXT_ALIGN_CENTER) drawText(10, 10, "Testtext") update()	Textausrichtung anpassen (ausgehend von der Mitte) TEXT_ALIGN_LEFT/CENTER/RIGHT
Text drehen 360 Grad -> 2 mal Pi Pi  Pi * 2 Pi/2	textFont(FONT_ROBOTO_48) translate(230,0) rotate(1.75) oder (PI/2) rotateText(true) drawText(0,0,"Hallo") drawText(0,50,"Welt") update()	Schriftgröße festlegen Nullpunkt verändern Rotation um 90 Grad Textdrehen aktivieren Erste Zeile Text Zweite Zeile text Textausgabe ausführen
Texteingabe	str = textInput("Name", "") drawText(10,10,str) update()	öffnet ein Text-Eingabefeld und gibt den Text zurück Test = Name der Eingabemaske "" = evtl. vordefinierte Eingabe
Ausgabe von Zeichen	drawChar(10,10,'?') update()	zeichnet ein Zeichen, wichtig –> einfache Hochkomma
Bildschirm löschen	clear()	löscht den Bildschirm
Hintergrundfarbe	background(255,0,255)	R,G,B zwischen 0 und 255
Funktionen		

Ereignisfunktion onDraw()
Aktualisierung des Bildschirms alle 20 ms für flüssige Bildschirmanimationen
	a = 10
	def onDraw()
	  drawRectangle(a,a,100,100)
	  update()
	  a = a + 1
		Variable a wird definiert
		Start onDraw-Funktion
		Rechteck mit x,y-Wert Variable a
		Darstellung des Rechtecks
		Erhöhung der Variable um 1

Ereignisfunktion onClick() Wird aufgerufen, wenn eine Joysticktaste gedrückt wird	def onClick() print(„Joystick gedrueckt“)	Start onClick-funktion Ausgabe vom entsprechenden Text
Ereignisfunktion onTimer() Sobald ein Timer abläuft, wird die Funktion aufgerufen	print("Text in 3 Sek") setInterval(3000) def onTimer(): print("Text")	Ankündigung Ruft das TimeEvent auf (in ms) Start TimeEvent Funktion im TimeEvent

Einfache Funktion	def rechne()
	  a = 10
	  b = 20
	  print(a+b)
	rechne()
		Funktionsdefinition
		Variable a
		Variable b
		Ausgabe der Addition
		Start der Funktion

Funktion mit Werteübergabe	def rechne(a,b) print(a+b) print(rechne(10,20))	Funktionsdefinition Ausgabe der Addition Start der Funktion mit Wertübergabe

Funktion mit Rückgabewert	def diff(a:int,b:int)->int:
	  if a>b:
	  return a-b
	  else
	  return b+a
	print(diff(3,4))
		Funktionsdefinition mit Datentyp
		Abfrage ob a größer b
		dann Rückgabe a – b (hier 1)
		wenn nicht
		dann Rückgabe b + a (hier 7)
		Aufruf Funktion und Ausgabe Wert

Buttonabfrage		

Buttonabfrage 1
getbuttons
	def onClick()
	  b = getButtons()
	  if b.up:
	  delay(500)
	  print(“Button hoch”)
		
		Übergabe d. Abfrage auf „b“
		Wenn b hoch
		Pause gegen entprellen
		up, down, left, right, middle

Buttonabfrage 2
buttons dient als Rückgabetyp von getButtons()
	def onDraw():
	  print(getButton())
	  update()
	  delay(1000)
		Es wird jeweils ein Button abgefragt 0 = kein Button;
		1 = Mitte; 2 = Hoch 3 = Rechts;
		4 = Runter; 5 = Links

Kontrollstrukturen		

Endlosschleife	while true:
	  y = random(0,240)
	  drawCircle(120,y,50)
	  update()
	  delay(100)
		solange wahr -> führe aus
		Zufallszahl
		Kreis m. zufälliger y-Position
		Zeigen
		Pause 0,1 Sek

Zählschleife	for i in [1..10]: print(i)	i = Zähler und […] Zählerbereich Schleife wird 11 mal durchlaufen Anzeige -> 0 bis 10
Zählschleife vereinfachte Form	for i in 10 print(i)	Schleife beginnt immer mit 0 Schleife wird 10 mal durchlaufen Anzeige -> 0 bis 9

Bedingungen
(Kopfschleife)
== ist gleich
!= ist nicht gleich
> oder < ist größer o.kleiner
>= o. <= ist größer o.kleiner gleich
	test = 1
	if test <= 2:
	  print(“kleiner 2")
	elif test == 2:
	  print(“Gleich 2)
	else:
	  print("groesser 2")
		Variable wird festgelegt
		Abfrage ob Variable kleinergleich 2
		dann Ausgabe -> kleiner
		Abfrage ob Variable gleich 2
		Ausgabe -> gleich 2
		ansonsten
		Ausgabe -> groesser 2

mehrere Bedingungen	if a >= 9 and <= 20 if b <= 8 or >= 5 if c > 0 not y==0	wahr nur wenn bei beiden wahr wahr wenn nur eine Bedingung wahr Aussage negiert durch not voran

while-Schleife
(ähnlich Zählschleife)
	i = 0
	while i < 5:
	  print(i)
	  i = i + 1
		Variable wird festgelegt
		Abfrage ob Variable kleiner 5
		Ausgabe der Variable (0 bis 4)
		Erhöhung der Variable um 1

Mathe-Funktionen		
Grundfunktionen	print(a + b) (a – b ; a * b ; a/b)	Division gibt nur Ganzzahl aus
Restwert der Division	print(10 % 3) #1	10/3 = 3 Rest 1
Runden	round(24.3) #24.0	rundet den Wert n auf den ganzzahligen Wert
Zufallszahl	x = random(0,240)	zufälligen Wert zw. min und max
Wertebereich umschreiben	const a = 25 # 0..10 v = map(a,0,10,0,255) print(v)	konvertiert den Wert eines Werte-bereichs in einen anderen (Wert a von 0 - 10 in Wert von 0 - 255)
Minimalwert	c = min(100,200) #100	liefert den tieferen Wert der Zahlen
Maximalwert	c = max(100,200) # 200	liefert den höheren Wert der Zahlen
Größte Ganzzahl	floor(203.8) # 203.0	gibt die größte ganze Zahl zurück, die kleiner oder gleich x ist
Kleinste Ganzzahl	ceil(203.8) # 204.0	gibt die kleinste Zahl zurück, die größer oder gleich x ist
Absoluter Wert	a = abs(-1) #1	berechnet den absoluten Wert a
Quadratzahl	Sqrt(9) #3	Berechnet die Quadratzahl
Kommazahl begrenzen	setPrecision(2)	Begrenzt alle Zahlen auf 2 Stellen

Zeichenfunktionen		
Bildschirm löschen	clear()	Löscht alle Pixel
Strich/Textfarbe festlegen	stroke(R,G,B)	setzt die Strich- oder Textfarbe Für RGB jeweils 0 - 255
Strichdicke festlegen	strokeWeight(x)	setzt die Strichdicke von 1 bis 10
Füllfarbe festlegen	fill(0,255,0) #Gelb	füllt das gezeichnete Objekt
Füllfarbe löschen	nofill()	Keine Füllfarbe, Durchsichtig
Nullpunkt verschieben	translate(120,120)	Verschiebt Nullpunkt relativ zur vorherigen Position
Linie zeichnen	drawLine(0,0,240,240)	xy= Anfang, x1y1=Ende Fenstergröße von 0 bis 240
Rechteck zeichnen 	drawRectangle(5,5,230,230)	xy=oben links, x1y1=unten rechts
Rechteck mit abgerun-deten Ecken zeichnen 	drawRectangleRounded(5,5,100,100,9)	zeichnet ein abgerundetes Rechteck, fünfter Wert= Radius
Dreieck zeichnen 	drawTriangle(5,5,100,5,50,100)	zeichnet ein Dreieck x0/y0, x1/y1, x2/y2
Viereck zeichnen 		zeichnet ein Viereck x0/y0, x1/y1, x2/y2, x3/y3
Kreis 	drawCircle(120,120,50)	Zeichnet einen Kreis x0/y0 mit Radius r
Ellipse zeichnen 	drawEllipse(100,100,70,90)	zeichnet eine Ellipse an die Pos x0/y0 und den Radien r0 und r1
Skaliert Zeichnungselement	scale(2)	skaliert alle nachfolgenden Zeich-nungsbefehle um den Wert x
Stilfarbe speichern speichert die vorhergehenden aktuellen Stil-Konfigurationen mit push() und stellt sie mit pop() wieder zur Verfügung	fill(255,0,0) push() fill(0,255,0) drawRectangle(0,0,100,100) pop() drawCircle(50,50,30) update()	Füllfarbe rot Zustand speichern Füllfarbe grün gr. Rechteck zeichnen alte Füllfarbe laden (aus fill) roten Kreis zeichnen Ausgabe
Zeit-Funktionen		
Objekt der Klasse	dt:dateTime	Objekt der Klasse (immer zuerst)
Aktuelle Zeit setzen	dt.now()	setzt das dateTime-Objekt gleich d. aktuellen o. eingestellten Uhrzeit
Anzeige akt. Jahr, Monat, Tag	getYear() / getMonth()/ getDay()	gibt aus: Jahr /Monat/Tag
Anzeige Stunde, Minute, Sek.	getHour() / getMinute() /getSecond()	gibt aus: Stunde /Minute /Sekunde
Anzeige des Wochentages	getWeekDay()	0=So, 1=Mo, 2=Di, 3=Mi, 4=Do, 5=Fr, 6=Sa
Datum ändern	setTime(23,59,55) # h, m, s	überschreibt die Zeitangabe
Zeit ändern	setDate(31,12,1990) # d, m, y	überschreibt das Datum
Timer	setTimer(3000) def onTimer(): print("Hallo")	ruft das onTimer()-Event nach xx ms auf
Zeitausgabe	t1 = getSystemTime() delay(5) t2 = getSystemTime() print(t2 – t1) # 4704 (kann variieren)	gibt die Zeit in Mikrosekunden seit dem Systemstart zurück. z.B. Um Zeit zw. Funktion zu messen
Pause	delay(1000)	Zeit in ms

String-Funktionen		
float zu int wandeln	print toInt(round(1.56)) #2	wandelt float in int-Typ
int zu float wandeln	print toFloat(120) #120.0	wandelt int-Zahl in float
string zu float wandeln	print strToFloat("120") #120.0	wandelt String in float
string zu int wandeln	print strToInt("120") #120	wandelt String in int
Oprationen		
Grundoperationen	+ , - , * , / , %(modulo oder Rest)	für Berechnungen
Vergleichsoperatoren	if a <= b: (== , < , > , =< , => , <= , !=)	Vergleich von Werten o. Zuständen
Stringvergleich	t1 = "test" print isEqual(t1,"test") # 1	Wenn beide strings Gleich sind wird true ausgegeben
Verknüpfungen	and , not , or if a = 3 and b = 2:	Verknüpfungen in Kontrollstrukturen
Binäre Funktionen bitweise Operation Bit verschieben	&(and), | (or), ^(xor) print (2 & 3) # 2 oder print (2 | 3) #3 <<, >> # Bit links bzw. rechts verschieben print(4<<2) # 16	and 010 (2) or 010 (2) 011 (3) 011 (3) 010 (2) 011 (3) Zahlen werden binär betrachtet 00100 -> 10000 #2mal nach links
Ausgabe Binärwert in Dez	print(0b01001) # 9	dazu wird 0b vorangestellt
Klassen und Objekte		
vector (bestehende Klasse) Variable, die einen zweidimen-sionalen Vektor speichern	v: vector(x=120,y=120) print(v.toString()) drawCircle(v.x,v.y,10) update()	Klasse vector erhält 2 float-Variablen Funktion von vector zur Anzeige Rechteck mit den Vectorwerten Anzeige vom Rechteck
dateTime (bestehende Klasse) die Klasse speichert einen Zeit-punkt, bestehend aus Datum und Zeit	dt:dateTime dt.setTime(22,33,44) print(dt.getHour()) print(dt.getMinute())	Objekt der Klasse erzeugen Setzen der Zeit (Std.,Min.,Sek.) Ausgabe der Stunde Ausgabe der Minute

color (bestehende Klasse)
Mit der Klasse color kann man Farben im RGB und HSV-Modus erzeugen.
	c = color()
	c.hsv(100,255,255) #oder
	# c.rgb(255,0,0)
	c.fill()
	drawCircle(120,120,30)
	update()
		Objekt der Klasse erzeugen
		Setzen der Farbe als HSV oder
		Setzen der Farbe als RGB
		für nachfolgendes Objekt
		Kreis zeichnen
		Anzeige Kreis

buttons (bestehende Klasse) Enthält die Funktion getButtons() die als Rückgabewert eine Variable vom Typ buttons liefert. up,down,left,right,middle	def onDraw() b = getButtons() if b.up: print("UP")	Dauerschleife Objekt der Klasse buttons Abfrage auf Objektvariable (bool) Ausgabe UP wenn wahr
Klasse erstellen (zuerst nur mit Variablen)	class Circle: x: int y: int radius: int c1:Circle(x=120,y=120,radius=30) clear() drawCircle(c1.x,c1.y,c1.radius) update()	Definition eigener Klasse Variable der Klasse benötigen einen Datentyp (Klassen- oder Objektvariablen) Deklarieren einer Variable Bildschirm löschen Kreis zeichnen mit Objektvariablen Kreis darstellen

Klasse erweitern
(erweitert um Funktionen)
	class Circle:
	x: int
	y: int
	radius: int
	def draw():
	drawCircle(x,y,radius)
	clear()
	c:Circle(x=120,y=120,radius=40)
	c.draw()
	update()
		Definition eigener Klasse
		Variable der Klasse benötigen
		einen Datentyp (Klassen oder Objektvariablen)
		Klassenfunktion anlegen
		Kreis zeichnen mit Klassenvariablen
		Bildschirm löschen
		Objekt der Klasse erzeugen
		Klassenfunktion aufrufen
		Kreis darstellen

MQTT

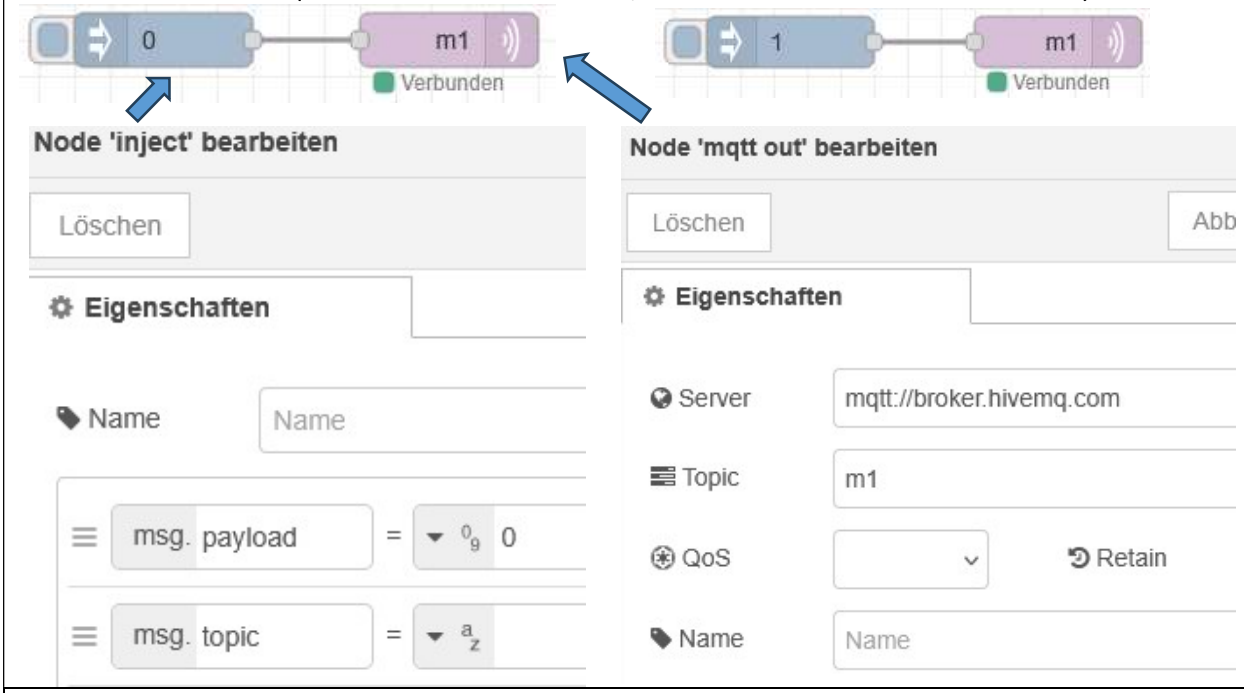
MQTT Empfang

Am Beispiel vom ein- und Ausschalten einer LED

```
initGPIO(C_PIN_05, OUTPUT)
uri = "mqtt://broker.hivemq.com"
username = ""
pass = ""
if connectMQTT(uri, username, pass):
   subscribeMQTT("m1")
def onDraw():
   if hasMQTTMessage():
      a = getMQTTData()
      writeGPIO(C_PIN_05,strToInt(a))
```

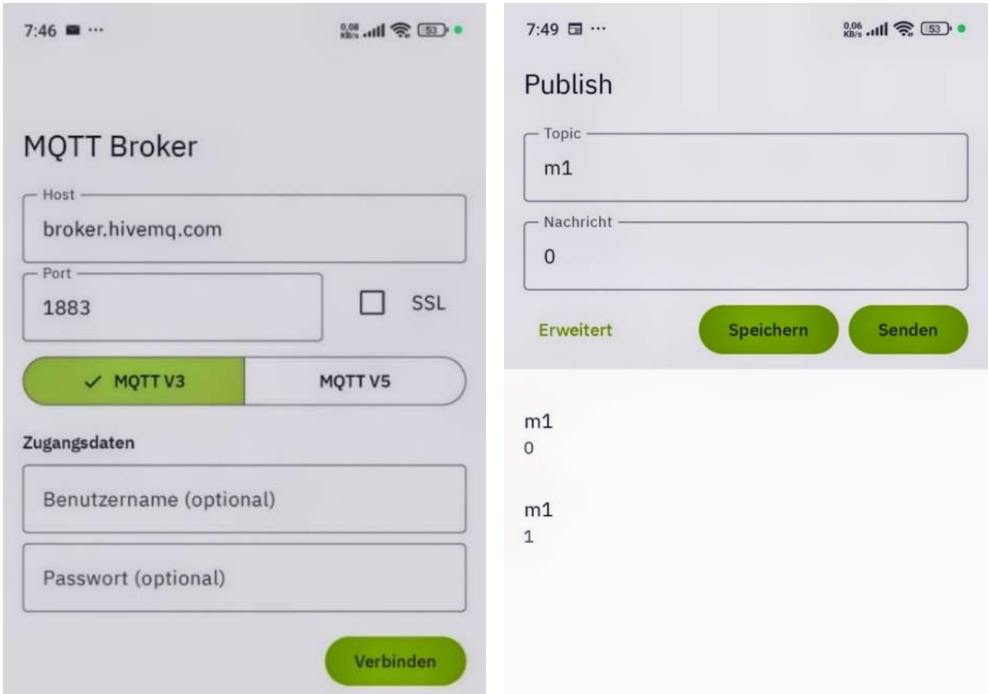
Initialisierung Port für LED
Adresse des MQTT-Broker´s
Benutzername
Passwort
wenn Verbindung und…
…Topic-Name
Dauerschleife
wenn Nachricht vom Broker
Nachricht an Variable a
bei 1 LED ein (bei 0 LED aus)

Senden mit Node-RED (hier Node-RED vom TXT 4.0; Aufruf im Browser: 192.168.x.x:1880) x.x = Adresse vom TXT



Senden mit Handy MyMQTT-App (als BeispielApp)

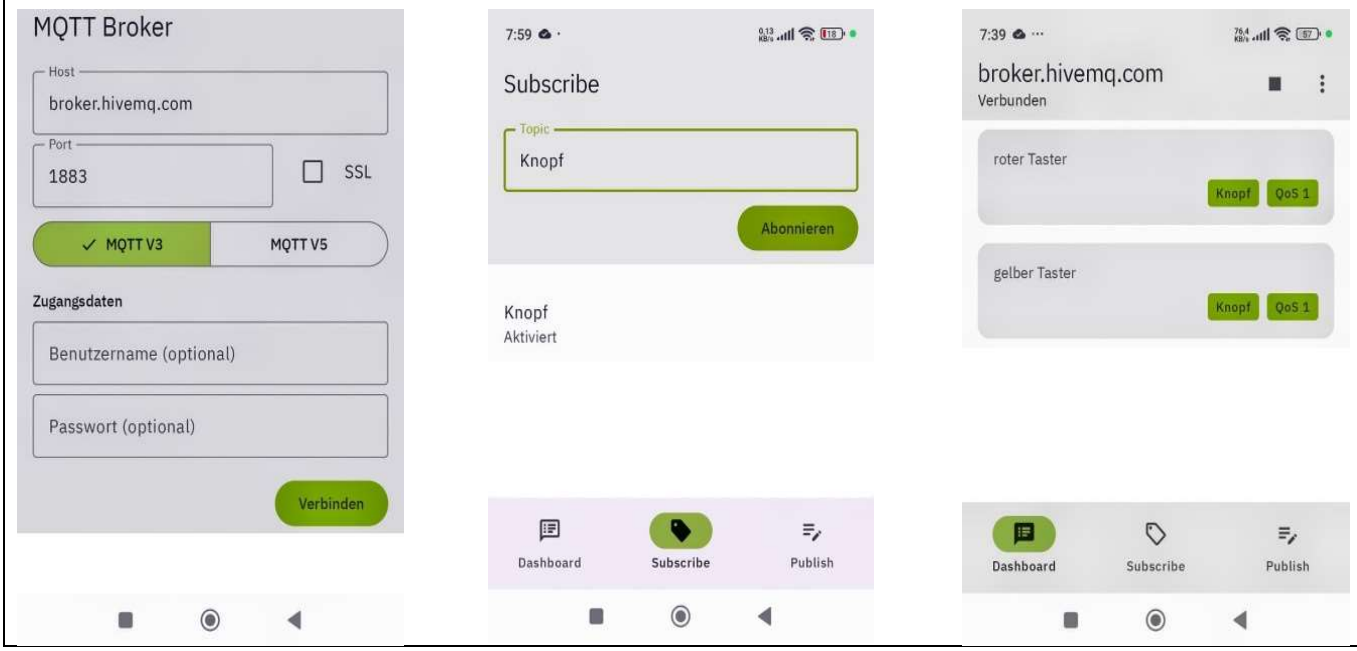
Das Nanopyscript ist dasselbe wie bei der Version von Node-Red
Bild 1 den MQTT-Broker eintragen und Verbinden
Bild 2 das Topic und die Nachricht eintragen und Senden



MQTT Senden
 	
Am Beispiel von zwei Button
	initGPIO(C_PIN_04, INPUT)
	initGPIO(C_PIN_05, INPUT)
	uri = "mqtt://broker.hivemq.com"
	username = ""
	password = ""
	def onDraw()
	  if readGPIO(C_PIN_04)
	  if connectMQTT(uri, username, password):
	  publishMQTT("Knopf", "gelber Taster")
	  delay(500)
	  if readGPIO(C_PIN_05)
	  if connectMQTT(uri, username, password):
	  publishMQTT("Knopf", "roter Taster")
	  delay(500)
		Initialisierung Port für Button PIN4
		Initialisierung Port für Button PIN5
		Variable für Broker
		Variable für Benutzername
		Variable für Passwort
		Dauerschleife
		Wenn Button Pin 4 gedrückt…
		und wenn Verbindung zum Broker
		Veröffentliche „gelber Taster“
		Pause um Taste zu entprellen
		Wenn Button Pin5 gedrückt…
		und wenn Verbindung zum Broker
		Veröffentliche „roter Taster“
		Pause um Taste zu entprellen

Empfangen mit Node-RED

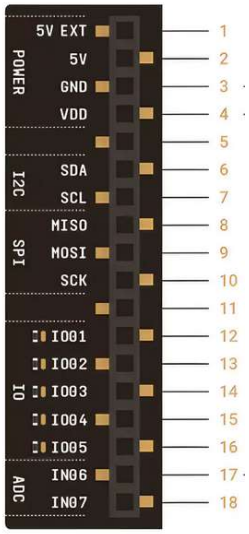
Empfangen mit HandyAPP



App mit Broker verbinden

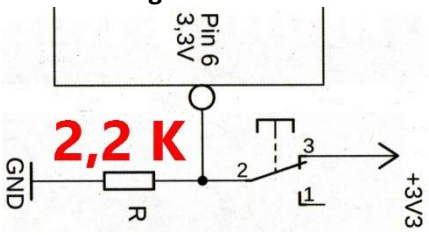
Name des zu überwachenden Topics festlegen

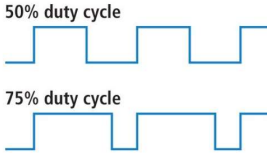
Anzeige der entsprechenden Nachricht

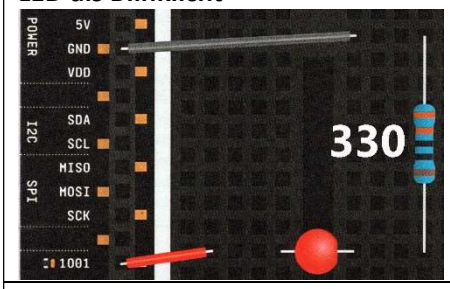
Hardware Aus- und Eingänge		
initGPIO 	initGPIO(pinNr.byte, mode:byte) C_PIN_01 (20*) Input/Output C_PIN_02 (25*) Input/Output C_PIN_03 (26*) Input/Output C_PIN_04 (32) Input/Output C_PIN_05 (33) Input/Output C_PIN_06 (34*) Input Analog C_PIN_07 (35*) Input Analog C_PIN_MISO (8) C_PIN_MOSI (5) C_PIN_SCK (7) C_PIN_SDA (21*) C_PIN_SCL (22*) **Ein-Ausgänge definieren** initGPIO(C_PIN_01, OUTPUT) initGPIO(20, OUTPUT)	Initialisiert das gewählte GPIO OUTPUT, INPUT, INPUT_PULLUP, INPUT_PULLDOWN *Diese GPIOs nur bei OXOcard-Connect Pin01 = OUTPUT bzw. INPUT Pin01 = OUTPUT (kurze Version)
Abfrage Digitalwert	if readGPIO(34): writeGPIO(C_PIN_05, 1)	Abfrage des als INPUT def. Pins wenn true -> LED an
Abfrage Analogwert	print(readADC(C_PIN_07, 100))	100 Messungen f. den Durchschnitt Wertebereich 0 bis 4095

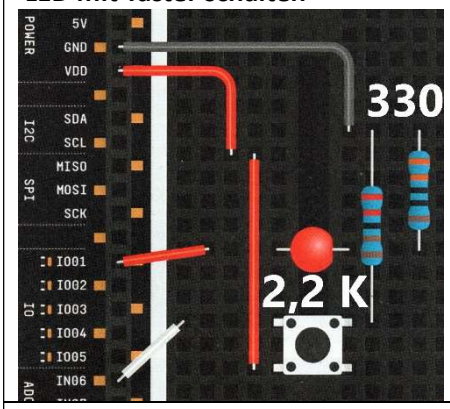
Ausgabe Digitalwert	writeGPIO(20, 1) #oder
	writeGPIO(C_PIN_01,1)
		20-verkürzte Angabe von Pin 1
		1 = ein, 0 = aus

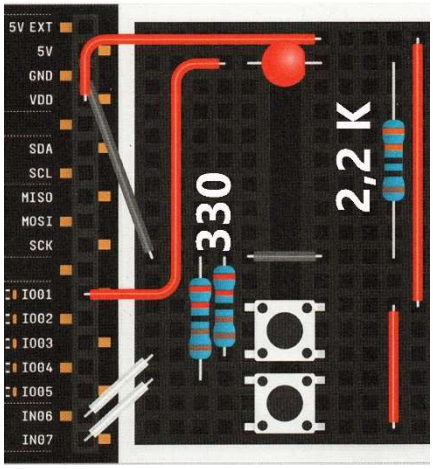
Ausgabe PWM-Signal	const DUTYCYCLE = 2048 # 0..4095 writePWM(C_PIN_01 , DUTYCYCLE)	Festlegung des Wertes (0-4096) Ausgabe des Wertes an Pin 1
Ausgabe Analogwert	const sp = 100 # 0..255 writeDAC(C_PIN_02, sp)	Festlegung des Wertes (0-255) Liest das Spannungssignal zwischen 0(0) und 3.3V(255) aus
Frequenz festlegen Es kann nur einmal eine Frequenz Im Programm vergeben werden	setPWMFrequency(Wert)	Töne = 523 bis 1046 Servo = 50 LED Helligkeit = 20 bis 500
Tonausgabe	setPWMFrequency(880) writePWM(C_PIN_05, 4096/2) delay(1000) writePWM(C_PIN_05, 0)	Frequenz wird festgelegt * Ausgabe mit 50% vom DutyCicle Tonausgabe 1 Sek. Ausgabe auf 0 (Ausschalten) *C-523,D-587,E-569,F-698 G-784, A-880,H-987,C´-1046
Helligkeit messen	print(readADC(C_PIN_06, 100))	Wert zwischen 0 und 4096
Temperatur messen	print(readADC(C_PIN_06,100))	Spannung am Spannungsteiler wird abgerufen (siehe eiter unten)
Bewegungssensor abfragen	if readGPIO(C_PIN_05): print("Alarm")	Wenn Digital-Pin HIGH wird hier Alarm angezeigt

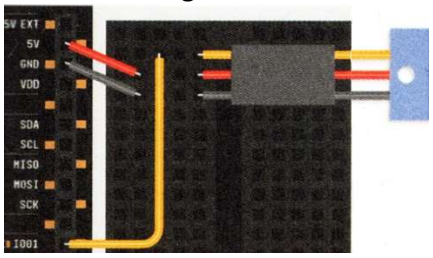
Buttonabfrage
 
	initGPIO(C_PIN_06, INPUT)
	initGPIO(C_PIN_05, OUTPUT)
	while true:
	if readGPIO(C_PIN_06):
	writeGPIO(C_PIN_05, 1)
	else:
	writeGPIO(C_PIN_05, 0)
	delay(10)
		
		Endlosschleife
		Abfrage ob Button gedrückt ist
		LED Pin 1 an
		Wenn nicht
		LED Pin 1 aus
		Pause 10 ms

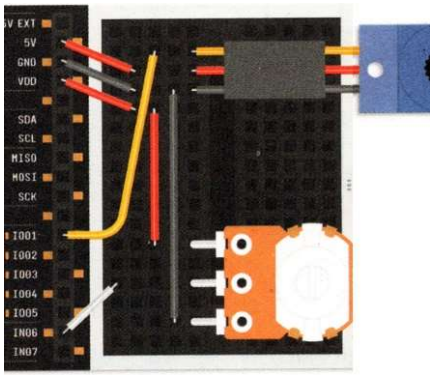
Helligkeit über PWM regeln
 
	const HELLIGKEIT = 2048 # 0..4096
	# const FREQUENCY = 500 # 20..500
	# setPWMFrequency(FREQUENCY)
	initGPIO(C_PIN_01, OUTPUT)
	writePWM(C_PIN_01 , HELLIGKEIT)
		Tastverhältnis An/Aus bei 500 Hz
		bei Bedarf Änderung der Frequenz
		hier wird neue Frequenz definiert
		Definition Pin1 als Ausgang
		Ausgabe PWM-Signal

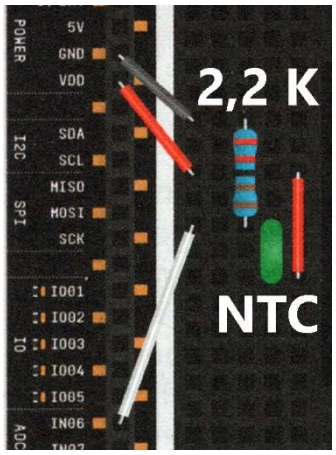
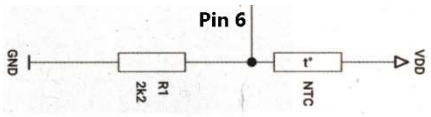
LED als Blinklicht
 
	const ZEIT = 500 # 1..1000
	initGPIO(C_PIN_01, OUTPUT)
	while true:
	  writeGPIO(C_PIN_01, 1)
	  delay(ZEIT)
	  writeGPIO(C_PIN_01, 0)
	  delay(ZEIT)
		Konstante zum regeln der Zeit
		Initialisierung Pin 1 als Ausgang
		Endlosschleife
		LED an
		Pause (siehe DELAY)
		LED aus
		Pause (siehe DELAY)

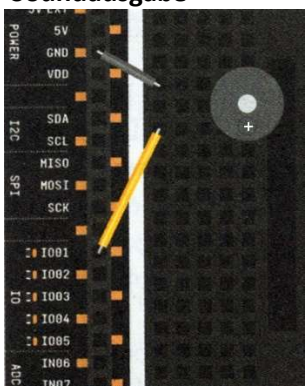
LED mit Taster schalten
 
	initGPIO(C_PIN_01, OUTPUT)
	initGPIO(C_PIN_06, INPUT)
	on = true
	while true:
	  button = readGPIO(C_PIN_06)
	  if button:
	  on = not on
	  delay(500)
	  writeGPIO(C_PIN_01, on)
		Pin 1 als Ausgang
		Pin 6 als Eingang
		Variable on wird auf true gesetzt
		Endlosschleife
		Einlesen Zustand Pin 6
		Wenn Button gedrückt
		Variable wird umgekehrt (true/false)
		Pause eine halbe Sekunde
		LED ein- oder ausschalten

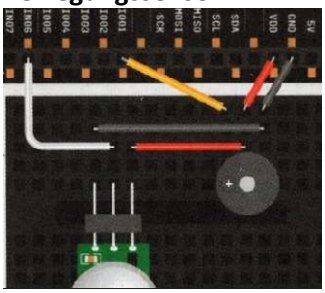
Helligkeitssteuerung mit 2 Tastern
 
	initGPIO(C_PIN_05, OUTPUT)
	initGPIO(C_PIN_06, INPUT)
	initGPIO(C_PIN_07, INPUT)
	setPWMFrequency(500)
	value = 50
	while true:
	  if readGPIO(C_PIN_06):
	  if value < 100:
	  value = value + 1
	  if readGPIO(C_PIN_07):
	  if value > 0
	  value = value -1
	  pwmValue = map(value,0,100,0,4096)
	  writePWM(C_PIN_05, pwmValue)
	  delay(10)
		LED-Pin (01) Ausgang
		Button (06)zum erhöhen
		Button (07)zum reduzieren
		Frequenz 50 Herz
		Aktuelle Helligkeit in Prozent
		Endlosschleife
		Button hoch gedrückt
		Ist Wert unter 100
		Wert um 1 erhöhen
		Button runter gedrückt
		Ist Wert über 0
		Wert um 1 reduzieren
		Wandeln v. Prozent in Wertbereich
		Wert über LED-Pin (01) ausgeben
		Pause 10 ms

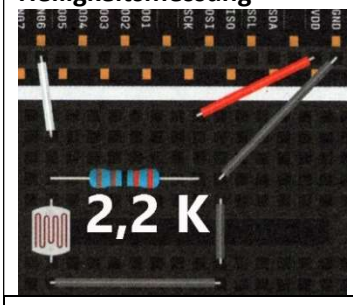
Servosteuerung
 
	setPWMFrequency(50)
	def setAngles(angle):
	  dutyCycle = map(angle, 90, -90, 102, 512)
	  writePWM(C_PIN_03, dutyCycle)
	setAngles(0)
	delay(1000)
	setAngles(-90)
	delay(1000)
	setAngles(90)
		50 Herz für Servosteuerung
		Funktion definieren
		Umwandlung Grad in Impulswert
		Ausgabe auf Pin 1 (Servo)
		Funktionsaufruf mit 0 Grad
		Pause 1 Sekunde
		Funktionsaufruf mit -90 Grad
		Pause 1 Sekunde
		Funktionsaufruf mit 90 Grad

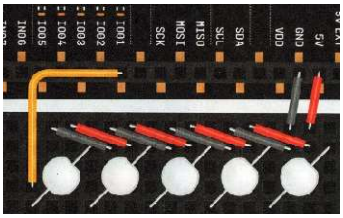
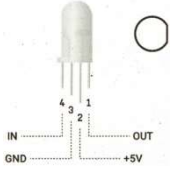
Servo mit Poti
 
	initGPIO(C_PIN_01, OUTPUT)
	initGPIO(C_PIN_07, INPUT)
	setPWMFrequency(50)
	while true:
	  a =(readADC(C_PIN_07, 100))
	  dutyCycle = map(a, 0, 4096, 102, 512)
	  writePWM(C_PIN_01, dutyCycle)
		Servo am Ausgang Pin1
		Poti zur Lenkung auf Pin7
		Frequenz für Servos
		Endlosschleife
		Analogwert von Poti auslesen
		Wandeln v. Potiwert in Impulswert
		Ansteuerung des Servos

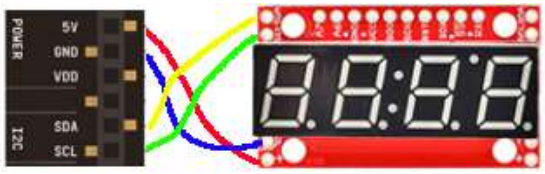
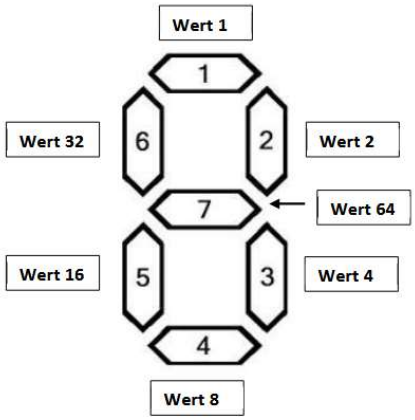
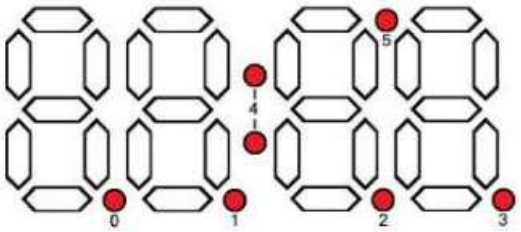
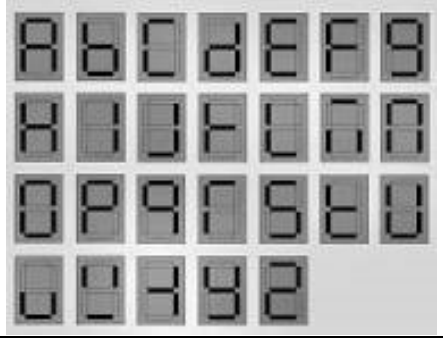
Temperaturmessung
 
 
	initGPIO(C_PIN_06, INPUT)
	const BETA = 4050.0
	const T25 = 298.15
	const R25 = 10000.0
	setPrecision(1)
	def drawDisplay(text:byte[])
	  clear()
	  textFont(FONT_ROBOTO_48)
	  drawTextCentered(120,120, text)
	  update()
	setInterval(100)
	def onTimer():
	  adcValue = readADC(C_PIN_06,100)
	  vr = 3.3 * adcValue /4095
	  vntc = 3.3 -vr
	  rntc = (vntc * 2200) / (3.3 - vntc)
	  tk = 1.0/(ln(rntc/R25)/BETA + (1.0/T25))
	  t = tk - 273.15
	  drawDisplay(t)
		Pin 7 als Analogeingang
		Beta-Wert des NTC (Datenblatt)
		Referenztemperatur
		Ref. Widerstand des NTC bei 25°
		Eine Nachkommastelle ausgeben
		Funktion für Textausgabe
		Alles löschen
		Textform
		Textausgabe mit Position
		Ausgabe starten
		ruft das onTime()Event nach ms auf
		Start Event-Timer
		Mittelwert aus 100 Messungen
		Spannung messen
		Spannung des NTC
		Widerstand NTC berechnen
		Temp. In Kelvin berechnen
		Umwandlung in Celsius
		Ausgabe auf Display

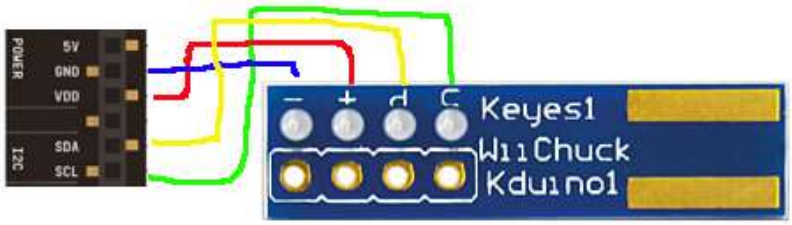
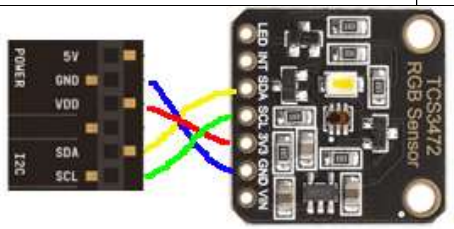
Soundausgabe
 
	const TONE_LENGHT = 200 #10..500
	const TONE_PAUSE = 50 # 50..500
	freq:int[8] =
	[523,587,659,698,784,880,987,1046]
	melody:int[8] = [1,2,1,2,1,2,3,4]
	for m in melody:
	  t = freq[m]
	  setPWMFrequency(t)
	  writePWM(C_PIN_05, 4096/2)
	  delay(TONE_LENGHT)
	  writePWM(C_PIN_05,0)
	  delay(TONE_PAUSE)
		Länge des Tons in ms |Änderung als
		Länge der Pause |Konstante
		Frequenzliste der 5.Oktave
		
		Liste von Tönen
		Melodieschleife
		Frequenz aus Liste holen
		Frequenz setzen
		DutyCicle von 50%
		Pause Tonlänge
		Ausschalten
		Pause

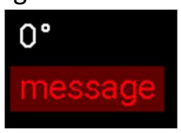
Bewegungssensor
 
	initGPIO(C_PIN_06, OUTPUT)
	initGPIO(C_PIN_01, INPUT)
	def onDraw():
	  if readGPIO(C_PIN_06):
	  print("Alarm")
	  writePWM(C_PIN_01,4096/2)
	  delay(500)
	  writePWM(C_PIN_01,0)
		Initialisierung Pin 06 (Sensor)
		Initialisierung Pin 01 (Sound)
		
		Abfrage ob Bewegung erkannt
		Ausgabe “Alarm” auf Terminal
		Ausgabe Alarmton
		Pause 500 ms
		Alarmton abschalten
		Sensor hält Info ca. 3 Sekunden

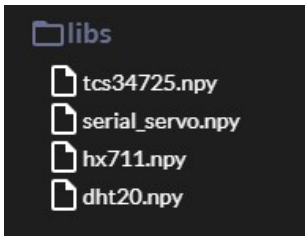
Helligkeitsmessung
 
	setPrecision(2)
	def drawDisplay(text:byte[]):
	  clear()
	  drawTextCentered(120,120, text)
	  update()
	while true:
	  adcValue = readADC(C_PIN_06, 100)
	  vldr:float = 3.3 * adcValue /4095
	  drawDisplay(vldr)
		2 Nachkommastellen
		Funktion für Textausgabe
		Alles löschen
		Text zentrieren
		Ausgabe starten
		Dauerschleife
		Mittelwert aus 100 Messungen
		Spannung berechnen
		Ausgabe der Spannung

WS-LED 	initDigitalLeds(C_PIN_04,2,C_LED_TYPE_WS2812) setDigitalLed(0, 255,0,0) setDigitalLed(1, 255,255,0) applyDigitalLeds() c:color c.hsv(200,200,20) setDigitalLed(0, c.r,c.g,c.b) 	Initialisierung für 2 LEDs an Pin 01 RGB-Farbe für LED 1 (0) RGB-Farbe für LED 2 (1) Ausgabe der LED-Daten Version mit hsv f. Helligkeitsregelung Farbwert,Sättigung,Helligkeit

I²C		
Test auf Vorhandensein eines Gerätes unter der Adresse xx	print(checkI2CAddress(0x20))	gibt true zurück, wenn ein Gerät mit definierter Addresse gefunden wurde
IO Board PCF8574 (0x20) (Porterweiterung Ein- und Ausgang)		
Datenbyte lesen	print(readI2CByte(0x20, 0x00)) #Geräteadresse, Datenwert	Anzeige des entsprechenden Wertes von dem Pin, an den 5V angelegt wurden
Datenbyte schreiben	writeI2CByte(0x20, 0x00,0x08) #Geräteadresse, Unteradresse, Datenwert	Am Ausgang von Pin4 (bin-Wert 08) wird eine Spannung von 3,5 V ausgegeben
LED Display COM-11440 (0x71) (4 * 7 Segmenteanzeige)		
Display löschen	writeI2CByte(0x71, 0x00, 0x76)	
Helligkeit	writeI2CByte(0x71, 0x7a, 0xff)	Helligkeit 0 – 255 (0x0 – 0xFF)
Linke Anzeige	writeI2CByte(0x71, 0x7b, 0x7f)	
2. Anzeige	writeI2CByte(0x71, 0x7c, 0x7f)	
3. Anzeige	writeI2CByte(0x71, 0x7d, 0x7f)	
Rechte Anzeige	writeI2CByte(0x71, 0x7e, 0x7f) Hex 7f = 127 (Addition aller 7 Segmente)	
Anzeige Punkte	writeI2CByte(0x71, 0x77, 0x10) Pos 0 = 0x01 Pos 1 = 0x02 Pos 2 = 0x04 Pos 3 = 0x08 Pos 4 = 0x10 Pos 5 = 0x20	
Anzeige der Ziffern Ziffer 1 = 0x06 | Ziffer 6 = 0x7D Ziffer 2 = 0x5B | Ziffer 7 = 0x07 Ziffer 3 = 0x4F | Ziffer 8 = 0x7F Ziffer 4 = 0x66 | Ziffer 9 = 0x6F Ziffer 5 = 0x6D | Ziffer 0 = 0x3F	Anzeige Alphabet Beispiel: S = 0x6d T = 0x54 O = 0x3f P = 0x50	

Nunchuck (0x52)		
Initialisierung 1 Handshake-Signal 1 Register	writeI2CByte(0x52, 0xf0, 0x55)	 Auswertung aller Werte auf dem Display
Initialisierung 2 Handshake-Signal 2 Register	writeI2CByte(0x52, 0xfb, 0x00)	
Beginn Dauerschleife	while true:	
Abfrage Rücksetzen	writeI2CByte(0x52, 0x00, 0x00)	
Notwendige Pause	delay(1)	
Joystick X-Achse	print(readI2CByte(0x52, 0x00))	Wert zwischen 0 und 255 Mittelstellung (nicht Bewegt) 128
Joystick Y-Achse	print(readI2CByte(0x52, 0x01))	Wert zwischen 0 und 255 Mittelstellung (nicht Bewegt) 128
Bewegung X	print(readI2CByte(0x52, 0x02))	Wert (Test) 75 bis 180
Bewegung Y	print(readI2CByte(0x52, 0x03))	Wert (Test) 75 bis 180
Bewegung Z	print(readI2CByte(0x52, 0x04))	Wert (Test) 75 bis 180
Abfrage C und Z Taste	print(readI2CByte(0x52, 0x05))	C und Z nicht gerückt=3; C und Z gedrückt = 0 C gerückt = 1; Z gedrückt = 2
Farbsensor TCS34725 (0x29) Ein Farbsensor misst die Frequenz-Bereiche des vom Objekt reflek-tierten Lichts.	 Die gemessenen Werte sind stark abhängig vom Umgebungslicht. Deshalb muss die Auswertung jeweils individuell angepasst werden.	
Initialisierung 1	writeI2CByte(0x29, 0x80, 0x00)	Kommandobit – Bit 7 auf 1 ist Pflicht bei allen Kommandoregister-Zugriffen
Initialisierung 2	writeI2CByte(0x29, 0x80, 0x03)	Enable-Register zum Einschalten - Power ON(PON)=0x01, AEN=0x02 -> 0x03
Beginn Dauerschleife für dauerhafte Abfrage	while true:	
Helligkeitswert	print(readI2CByte(0x29, 0x94))	Wert der Helligkeit (max. 255)
Wert der Farbe Rot	print(readI2CByte(0x29, 0x96))	Wert der Farbe (max. 255)
Wert der Farbe Grün	print(readI2CByte(0x29, 0x98))	Wert der Farbe (max. 255)
Wert der Farbe Blau	print(readI2CByte(0x29, 0x9a))	Wert der Farbe (max. 255)
	delay(1000)	

Bibliotheken		
IO Vereinfachung der Ansteuerung von Hardware (bereits integriert)	import io def onDraw(): IO.writePin(C_PIN_05, 1) delay(500) IO.writePin(C_PIN_05, 0) delay(500)	Import der Bibliothek Endlosschleife Setzt das IO auf 1 Pause 0,5 Sekunden Setzt das IO auf 0 Pause 0,5 Sekunden
Monitor Systemmeldungen auf den Screen ausgeben (bereits integriert) 	import monitor	Import der Bibliothek
	monitor.init()	Initialisierung
	monitor.push("0°")	Ausgabe einer Meldung
	monitor.pushc("message",MONITOR_RED)	Ausgabe einer Meldung in Farbe _RED, _GREEN,_ YELLOW, _BLUE
serielle Servos -eigener Controller -größerer Winkel (300°) -bidirektional, Rückgabe vom Winkel, -253 Servos am Bus	import serial_servo	Import der Bibliothek
	servoBus:SerialServoBus	Deklaration der Variable
	servoBus.init(C_PIN_02,C_PIN_01,1000000)	Initialisierung; Anschlüsse des UART_Transceivert tx = Pin 02; rx = Pin 01, Speed = 1.000.000 VS an 5V !! und VDD an 3,3 V !!
	servoBus.setPosition(1,500,1000)	Bewegungsbefehl; 1 = Servonr. Zielwinkel zwischen 0 u.1000 Geschwindigkeit (0 bis 1000)
	servoBus.stopServo(1) servoBus.stopAllServos()	Servo deaktivieren – Servo lässt sich nun manuell bewegen
Positionsrückgabe	print(servoBus.getPosition(1))	Rückgabe der Position
Servo als Motor	servoBus.setMotorSpeed(1,200,true) servoBus.setAllMotorSpeeds (200,true)	Servonr., Speed 0 - 1000, Drehrichtung true = re. false = li.
ID ändern	servoBus.changeId(1,2)	alte ID dann neue ID; für z.B.Bus
Servo erkennen	servoBus.isServoMoving(2)	Anzeige ob Motor erkannt
hx711 für Wägezelle	import hx711	Import der Bibliothek
	myScale:HX711	deklarieren der Variable
	myScale.init(C_PIN_02, C_PIN_01)	Initialisierung; Anschlüsse des UART tx = Pin 02; rx = Pin 01
	mySale.tare()	Waage auf 0 setzen
	print(myScale.getWeight())	Ausgabe des Gewichtes
eigene Bibliothek Code schreiben und unter z.B. „myLib“ im Ordner libs speichern	def printText(text:byte[]) clear() drawTextCentered(120,120,text) update()	Funktion definieren Bildschirm löchen Ausgabe vorbereiten Text ausgeben
Benutzung der Biblothek	import myLib printText("Hallo")	In neuer Datei Bibl. Aufrufen Verwendung der Funktion



Außer den Bibliotheken IO und Monitor, müssen die anderen in das Verzeichnis „libs“ kopiert werden.