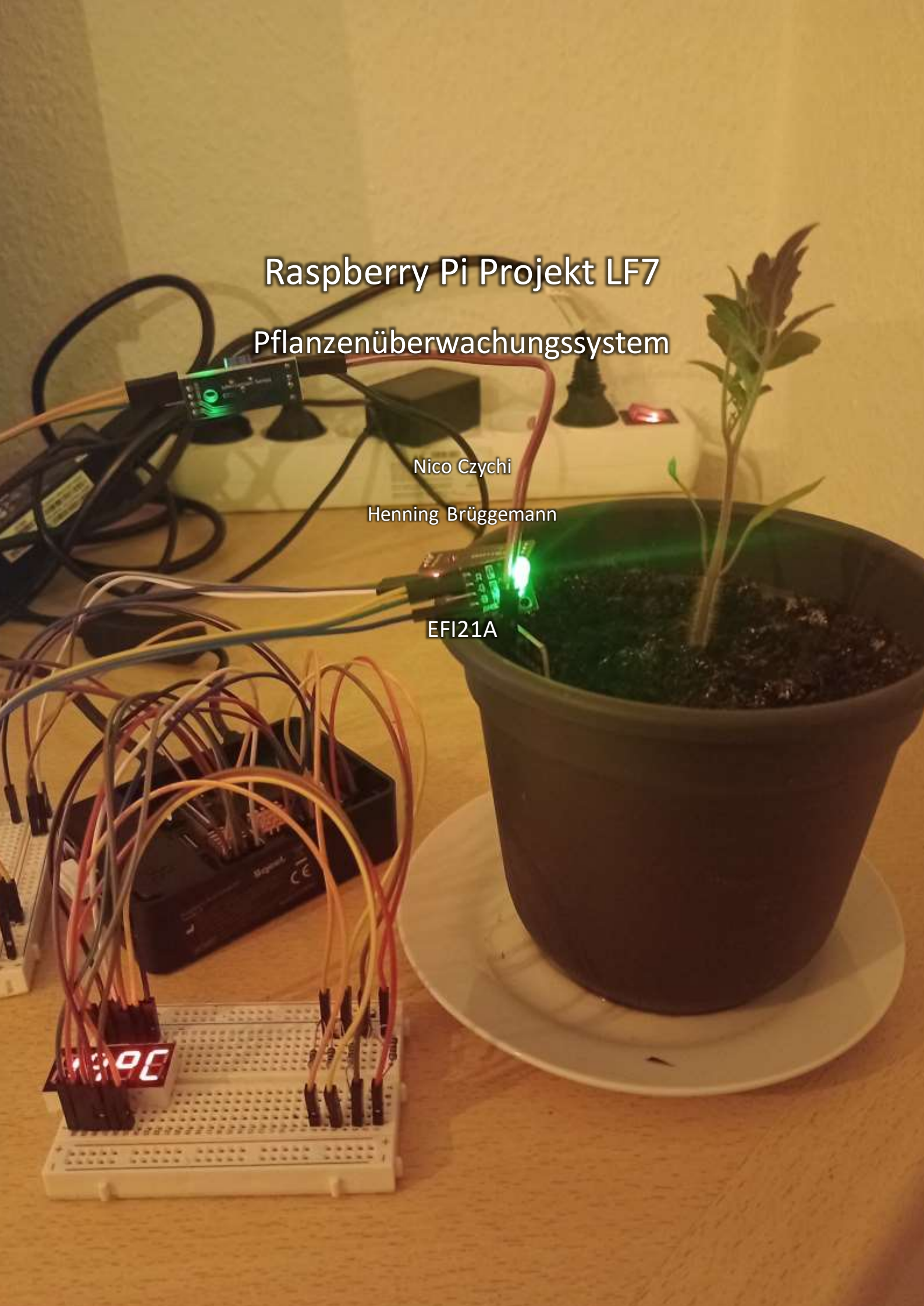


Raspberry Pi Projekt LF7

Pflanzenüberwachungssystem

Nico Czychi

Henning Brüggemann

EFI21A

Inhaltsverzeichnis

Allgemeine Funktionsweise

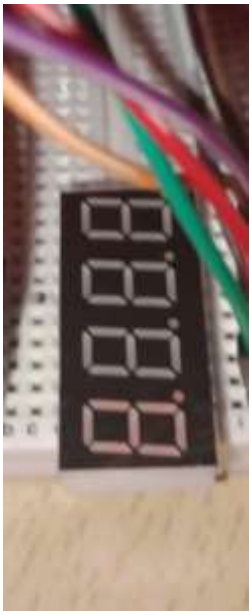
Das Pflanzenüberwachungssystem soll eine Pflanze und deren Umgebung überwachen, um eine optimale Umgebung zum Wachsen und Erhalten der Pflanze zu gewährleisten. Dazu misst das System die Umgebungstemperatur und gibt diese für den Hobbybotaniker gut sichtbar auf einer LED-Anzeige aus. Die Temperaturangabe wird dabei alle 10 Sekunden aktualisiert. Neben der Umgebungstemperatur ist auch die Bodenfeuchtigkeit ein entscheidender Faktor für die Pflanzen. Damit der Hobbybotaniker nicht vergisst seine Pflanzen zu gießen oder um zu vermeiden, dass die Pflanze zu viel gegossen wird, misst das System die Bodenfeuchtigkeit und lässt eine LED rot leuchten, wenn gegossen werden muss und grün, wenn die Pflanze kein weiteres Wasser benötigt. Der Bodenfeuchtigkeitssensor lässt sich dabei über ein Drehrad einstellen.

Mit diesem System sollte es allen Hobbygärtnern möglich sein, den Pflanzen in der Wohnung ein langes und gesundes Leben zu ermöglichen.

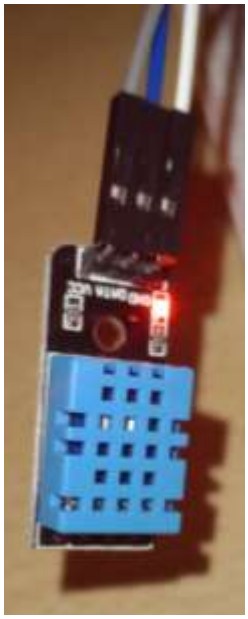
Eingesetzte Aktoren und Sensoren

Die Aktoren des Systems sind zum einen eine LED, welche grün oder rot leuchtet und eine 4fach 7-Segment Anzeige, welche die Temperatur in °C ausgibt. Die Sensoren sind ein DHT11 Temperatur und Luftfeuchtigkeitssensor und ein Bodenfeuchtigkeitssensor.

RGB LED



4fach 7-Segment Anzeige



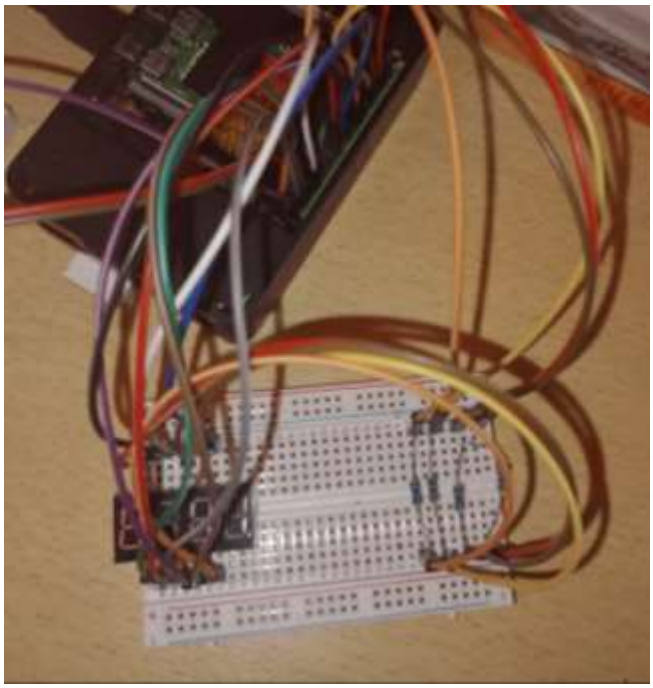
DHT11 Temperatur-/Luftfeuchtigkeitssensor

Bodenfeuchtigkeits-sensor

Verkabelung

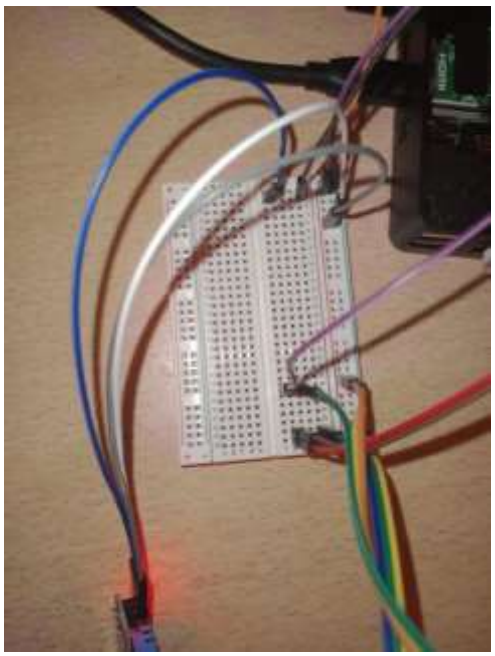
Für die Verkabelung wurden 2 Mini-Breadboards und ein Raspberry Pi verwendet. Die 2 Mini-Breadboards sorgen für mehr Platz bei der Verkabelung, da die 4fach 7-Segment Anzeige alleine schon sehr viele Kabel benötigt.

Auf dem ersten Mini-Breadboard befindet sich die 4fach 7-Segment Anzeige. Hierbei müssen alle Striche einer Ziffer, sowie deren Punkt je mit einem Kabel an einen Pin des Raspberry Pi angeschlossen werden. Des Weiteren werden je ein Kabel für die Ansteuerung der einzelnen Ziffern benötigt, welche über 1000 Ohm Widerstände an je einen Pin des Raspberry Pi angeschlossen werden.



Verkabelung der 4fach 7-Segment Anzeige

Auf dem zweiten Mini-Breadboard werden die 2 Sensoren und die LED verkabelt. Für den Bodenfeuchtigkeitssensor, wurde sich dafür entschieden nur das digitale Signal zu verkabeln und auszuwerten, da der Pi von Hause aus keine Analogen Signale verarbeiten kann. Bei der LED wurden nur die Anschlüsse für Rot und Grün verkabelt, da nur diese beiden Farben benötigt werden. Der Temperatur- und Luftfeuchtigkeitssensor wurde komplett angeschlossen.



Verkabelung der Sensoren und der LED

Quellcode

Der Quellcode wurde in Python geschrieben. Zunächst werden die benötigten Bibliotheken importiert.

```
1   # Importieren der benötigten Bibliotheken
2   import sys
3   import RPi.GPIO as GPIO
4   import Adafruit_DHT
5   import time
```

Danach wird der Modus zum ansprechen der Pins definiert. In diesem Fall der BCM-Modus, bei dem die Pins über ihren definierten Namen und nicht über ihre Position angesprochen werden. Möchte man eine Übersicht über diese Namen erhalten reicht der Befehl „pinout“ im Terminal.

```
9   # Benutzen der Pin Namen statt ihrer physischen Anordnung
10  GPIO.setmode(GPIO.BCM)
11
```

Definieren der Pins

Anschließend werden die Pins für den Temperatursensor (= 4), den Bodenfeuchtigkeitssensor definiert (= 21) und der LED (= 6, 19) und deren Funktion als Output- oder Input-Pin.

```
12  # Temperatursensor
13  DHT_SENSOR = Adafruit_DHT.DHT11
14  DHT_PIN = 4
15
16  # Bodenfeuchtigkeitssensor
17  data = 21
18  green = 6
19  red = 19
20
21  GPIO.setup(data, GPIO.IN)
22  GPIO.setup(red, GPIO.OUT)
23  GPIO.setup(green, GPIO.OUT)
24
25  GPIO.output(green,GPIO.LOW)
26  GPIO.output(red,GPIO.LOW)
27
```

Für die 4fach 7-Segment Anzeige wird zunächst die Zeit definiert, wie lange eine einzelne Ziffer der Anzeige leuchten soll. Dies ist notwendig, weil immer nur eine Ziffer leuchten kann und durch eine kurz nacheinander geschaltete Folge der Ziffern eine Art optische Täuschung für das menschliche Auge entsteht und für den Betrachter der Anschein erweckt wird, als würden alle 4 Ziffern zur gleichen Zeit leuchten. Anschließend werden die Pins für die 4 Ziffern definiert (selDigit) und die Pins für die Segmente der Ziffern, also die Striche und der Punkt definiert (display_List). Diese Pins werden jeweils in ein Array gespeichert, was später die Ansprache und Schaltung der Pins vereinfacht. Des Weiteren wird auch ein Pin für den Punkt vergeben.

```
28  # Sieben-Segment-Anzeige
29
30  delay = 0.0005 # Zeitverzögerung zwischen den einzelnen Ziffern
31
32  # Ziffern:   1, 2, 3, 4
33  selDigit = [14,15,18,23]
34
35  #Segmente: A ,B ,C ,D ,E ,F ,G
36  display_list = [24,25,10,9,26,12,16]
37
38  #Punkt = GPIO 20
39  digitDP = 20
```

Anschließend werden über eine Schleife alle Pins des Arrays für die Ziffern und des Arrays für die Segmente als Output definiert. Um in der Konsole währenddessen kein Warnanzeigen zu erhalten, werden diese für diesen Prozess deaktiviert.

```
42  # Setze all Pins als Output
43  GPIO.setwarnings(False)
44  for pin in display_list:
45      GPIO.setup(pin,GPIO.OUT) # setzen der Pins für die Segmente
46  for pin in selDigit:
47      GPIO.setup(pin,GPIO.OUT) # setzen der Pins für die ziffern
48  GPIO.setup(digitDP,GPIO.OUT) #
49  GPIO.setwarnings(True)
```

Nach dem setzen der Pins wird ein mehrdimensionales Array mit Elementen gebildet, die ebenfalls Arrays sind und die jeweilige Schaltung der Segmente für jede Zahl sowie dem „°“-Zeichen und dem Buchstaben „C“ beinhalten. „0“ steht hierbei für „leuchtet“ und „1“ für „leuchtet nicht“. Das besondere hierbei ist, dass der Index der Elemente des mehrdimensionalen Arrays der passenden Zahl entspricht. Index 0 enthält somit die Schaltung für die Zahl 0 usw. Index 10 und 11 enthalten die Schaltung für das Grad-Zeichen sowie dem Buchstaben C. Zum Abschluss der Pin-Definitionen wird der Punkt-Pin auf „aus“ gestellt.

```
51  # DIGIT map as array of array ,
52  #so that arrSeg[0] shows 0, arrSeg[1] shows 1, etc
53  arrSeg = [[0,0,0,0,0,0,1],\
54            [1,0,0,1,1,1,1],\
55            [0,0,1,0,0,1,0],\
56            [0,0,0,0,1,1,0],\
57            [1,0,0,1,1,0,0],\
58            [0,1,0,0,1,0,0],\
59            [0,1,0,0,0,0,0],\
60            [0,0,0,1,1,1,1],\
61            [0,0,0,0,0,0,0],\
62            [0,0,0,0,1,0,0],\
63            [0,0,1,1,1,0,0],\
64            [0,1,1,0,0,0,1]]
65
66  GPIO.output(digitDP,0) # Punkt aus
67
68
```

Funktionen

Die GetTemperature()-Funktion liest die gemessenen Werte des Temperatursensors ein und wenn dieser ein Signal gibt, wird die Temperatur aufgerundet und in einen String umgewandelt und mit dem „°C“ Zeichen verknüpft. Sollte der Sensor kein Signal geben, wird nur 0000 übergeben. Sollte dies passieren, so sollte die Verkabelung überprüft werden.

```
71  # Auslesen des Temperatursensors
72  def getTemperature():
73      humidity, temperature = Adafruit_DHT.read(DHT_SENSOR, DHT_PIN) #Temperatur auslesen
74      if humidity is not None and temperature is not None:
75        temperature = str(round(temperature)) + "°C" #Aufrunden der Temperatur und verknüpfen mit '°C'
76      else:
77          temperature = "0000" #bei keinem Signal alles auf 0
78      return temperature
79
```

Die getMoistureData()-Funktion liest das Digitalsignal des Bodenfeuchtigkeitssensors aus und schaltet die LED in den entsprechenden Farben an. Gibt der Sensor eine 1 aus, so hat er keinen Wasserkontakt, bei einer 0 hat er Kontakt zu Wasser.

```
80  # Auslesen des Bodenfeuchtigkeitssensors und steuern der LED
81  def getMoistureData():
82      if GPIO.input(data):
83          GPIO.output(green, GPIO.LOW) #bei keinem Wasser geht Grün aus
84          GPIO.output(red, GPIO.HIGH) #und Rot leuchtet
85      else:
86          GPIO.output(green, GPIO.HIGH) #bei Wasser geht Grün an
87          GPIO.output(red, GPIO.LOW) #und Rot geht aus
```

Die showDisplay(digit)-Funktion nimmt die Zahl die ausgegeben werden soll auf und schaltet dementsprechend die Segmente und Ziffern an. Dazu wird ein Schleifendurchlauf für jede der 4 Ziffern gestartet. Bei jedem Schleifendurchlauf werden zunächst alle Ziffern auf „aus“ und anschließend nur die jeweilige Ziffer des Schleifendurchlaufs auf „an“ gestellt.

```
89  # Aktiviert die Segmente entsprechend der übergebenen Zahl
90  def showDisplay(digit):
91   for i in range(0, 4): #Schleifendurchlauf für alle Ziffern
92    sel = [0,0,0,0]
93    sel[i] = 1
94    GPIO.output(selDigit, sel) #aktivieren der ausgewählten Ziffer
```

Anschließend wird geprüft, ob es sich bei der jeweiligen Ziffer der übergebenen Zahl um ein Leerzeichen handelt. Sollte dem so sein, so werden alle Segmente ausgestellt und der Schleifendurchlauf beendet.

```
95    if digit[i].replace(".", "") == " ": #Leerzeichen deaktiviert die Ziffer
96     GPIO.output(display_list,0)
97     continue
```

Sollte die Ziffer kein Leerzeichen sein, so wird im nächsten Schritt geprüft, ob es sich um das °-Zeichen handelt. Sollte dem so sein, so wird der entsprechende Index aus dem Mapping-Array angesprochen, in diesem Fall Index 10. Sollte es sich um das „C“ handeln, so wird der Index 11 angesprochen.

```
98      if digit[i] == "°":
99       GPIO.output(display_list,arrSeg[10]) #bei Grad-Zeichen wird der Index 10 angesprochen
100      time.sleep(delay)
101      continue
102     if digit[i] == "C":
103      GPIO.output(display_list,arrSeg[11]) #bei C wird Index 11 angesprochen
104      time.sleep(delay)
105      continue
```

Sollten alle Bedingungen nicht zutreffen so wird die Ziffer als Index zum Ansteuern der Schaltung im Mapping-Array genutzt, da der Index von 0 bis 9 des Mapping-Arrays den entsprechenden Zahlen entspricht und somit die passenden Segmente angesteuert werden können.

```
106     numDisplay = int(digit[i].replace(".", ""))
107     GPIO.output(display_list, arrSeg[numDisplay]) # die Segmente werden dementsprechenden Mapping angeschaltet
```

Danach wird noch geprüft, ob es einen Punkt zu der Zahl gibt, bei dem ebenfalls der Punkt-Pin angesteuert wird. Somit wäre theoretisch auch die Ausgabe einer Kommazahl möglich. Am Ende eines jeden Schleifendurchlaufs wird die Zeit gewartet, die zuvor definiert wurde bis die nächste Ziffer leuchtet.

```
108     if digit[i].count(".") == 1:
109      GPIO.output(digitDP,0)
110     else:
111      GPIO.output(digitDP,1)
112     time.sleep(delay)
113
```

In der **Funktion splitToDisplay(toDisplay)** wird die übergebene Zahl für die weitere Verarbeitung aufbereitet. Dazu wird zunächst der übergebene String in ein Array umgewandelt wobei jedes Element eine Ziffer des Strings ist. Anschließend wird für jedes Element geprüft, ob es ein Punkt ist. Falls dem so ist, wird dieses Element dem vorangegangen zugeordnet da in der showDisplay() -Funktion genau diese Formatierung erwartet wird. Anschließend wird der Punkt als alleiniges Element aus dem Array gelöscht, da er der vorigen Zahl zugeordnet wurde.

```
114  # Aufbereiten der übergebenen Werte indem der String geteilt wird
115  def splitToDisplay (toDisplay):
116   arrToDisplay=list(toDisplay)
117   for i in range(len(arrToDisplay)):
118    if arrToDisplay[i] == ".": arrToDisplay[(i-1)] = arrToDisplay[(i-1)] + arrToDisplay[i] # Punkte werden mit dem vorigen Element verknüpft
119   while "." in arrToDisplay: arrToDisplay.remove(".") # Array Elemente mit Punkt werden gelöscht
120   return arrToDisplay
121
```

Main Loop – Ausführung des Programms

Im Main-Loop findet sich eine Endlos-Schleife, die zunächst alle Ziffern auf 0 setzt. Das geschieht deshalb, damit keine Ziffern aus dem vorherigen Durchlauf des Programmes leuchten. Anschließend wird ein Timer auf 10 Sekunden definiert in der die Temperatur ausgegeben werden soll., sowie die Bodenfeuchtigkeit gemessen wird. Beim abbrechen des Programms werden alle Ziffern wieder ausgestellt und die Einstellungen der Pins resettet.

```
123
124  ######## MAIN LOOP ########
125
126  try:
127    while True:
128      GPIO.output(selDigit, [0,0,0,0])
129      t_end = time.time() + 10 #definieren des Zeitintervalls zum Temperatur-Refresh
130      toDisplay = getTemperature()
131      while time.time() < t_end:
132        getMoistureData()
133        showDisplay(splitToDisplay(toDisplay))
134  except KeyboardInterrupt:
135    print('interrupted!')
136    GPIO.output(selDigit, [0,0,0,0])
137    GPIO.cleanup()
138  sys.exit()
139
```