Handreichung: Physical Computing

1. Kurzvorstellung

Dies ist der zweite Teil des Projektes „Smartes Gewächshaus“. Im ersten Teil (von Nils Beyer) geht es um die automatische Steuerung eines kleinen „Gewächshauses“ (realisierbar zB. in einem Blumenkasten auf einer Fensterbank des Informatikraums). In diesem Teil geht es schließlich um die automatische Überwachung dieses Gewächshauses inklusive Auswertung der Messdaten und Bereitstellung ebendieser in einem Cloudspeicher.

Zusammen ergeben diese beiden Bausteine genug Material für das „Informatische Projekt“ in der Oberstufe des sächsischen Lehrplans. Die Konzeption richtet sich dabei vom Umfang vorrangig an Leistungskurse. Für Grundkurse lässt sich das Projekt aber adaptieren, indem entsprechend mehr von der Lehrkraft vorgegeben wird und zB. einzelne Klassen des Programmcodes zur Verfügung gestellt werden.

Dabei wird das Gewächshaus vorrangig über einen Arduino gesteuert und verwaltet (siehe Handreichung von Nils Beyer). Anschließend werden die Daten auf einen Raspberry Pi übertragen und von diesem ausgewertet und aufbereitete Plots der Messdaten in eine Cloud geladen.

Die Projektarbeit soll dabei vom eigenständigen Aufteilen der Schüler:innen in Kleingruppen zum Bearbeiten von Teilaufgaben geprägt sein, nachdem gemeinsam ein Plan für das Aufteilen erstellt wurde (unter anderem das gemeinsame Erstellen des UML-Diagramms). Teil des Prozesses sind dabei wiederkehrende Zwischenmeetings zum Evaluieren der Arbeitsstände und anpassen der Teilgruppen. Zur Bewertung bietet sich das Verfassen eines Projektberichtes von jeder Schüler:in an … hierdurch tritt das gemeinsame Ergebnis in der Bewertung in den Hintergrund und die Schüler:innen sind gezwungen, sich über die Arbeitsschritte der jeweils anderen Kleingruppen auszutauschen. Der Anteil der jeweiligen Schüler:innen am Projekt kann natürlich mit in die Bewertung einfließen, sollte aber nicht den Großteil ausmachen …

Nach einer kurzen Vorstellung der Rahmens für die nächsten Unterrichtswochen und des vorhandenen Materials (inklusive Nennung nützlicher Python-Bibliotheken, siehe unten) schlüpft die Lehrkraft in die Rolle des Kunden, welcher eine Realisierung für seine Gewächshausidee von den Entwicklern (dem Leistungskurs) haben will. Dabei wird kein fertiger Arbeitsauftrag ausgeteilt, sondern die Schüler:innen sollen sich durch Nachfragen einen eigenständigen Überblick über das erwünschte Produkt verschaffen und anschließend in die Planungsphase übergehen. Die Lehrkraft kann hier natürlich außerhalb ihrer Rolle als Kunde weitere Tipps und Hinweise geben.

2. Einordnung in die Lehrpläne

Klasse	OS	Gy	BS
7			
8			
9			
10			
11/12		Informatisches Projekt (GK: LB 7, LK: LB 11)	

3. Lernziele

Kognitiv

Die Schüler:innen erstellen zusammen einen Ablaufplan für das Informatische Projekt.

Die Schüler:innen zerlegen das Informatische Projekt gemeinsam in kleinere Teilaufgaben, die unabhängig voneinander gelöst und implementiert werden können.

Die Schüler:innen evaluieren in „Zwischenmeetings“ ihre Arbeitsstände, um die Projektteile am Ende sinnvoll zu einem Ganzen zusammensetzen zu können.

Die Schüler:innen übernehmen in kleineren Teams die Verantwortung für einzelne Teilbereiche des Projektes. Sie implementieren und testen ihre Teilprogramme.

Affektiv:

Die Schüler:innen organisieren sich eigenständig in den Teilaufgaben, entsprechend ihrer persönlichen Stärken und Schwächen.

Die Schüler:innen arbeiten kollaborativ an dem Informatischen Projekt zusammen uns nehmen aufeinander Rücksicht.

4. Voraussetzungen

Technische Voraussetzungen:

Blumenkasten, Arduino, Temperatursensor, Feuchtigkeitssensor (Bodenfeuchte), Pumpe, Raspberry Pi, Account in Cloud, Internetzugang

Im Cloudanbieter der persönlichen Wahl muss über die Entwickleroptionen eine App eingestellt werden, in welcher die erstellten Plots gespeichert werden können. Alternativ lassen sich die Plots auch in ein mit einer Cloud synchronisierten Ordner auf dem Laufwerk des Raspberry Pis legen. Bei dieser Lösung fällt die Synchronisation der Daten mit einer Cloud deutlich leichter aus, da die Authentifikation zum Start des Programms wegfällt. Hierdurch wird hier eine Schüler:innengruppe weniger benötigt.

Fachliche Voraussetzungen:

Das informatische Projekt ist im Lehrplan als letzter Lernbereich angesiedelt. Dies hat den Hintergrund, dass die Schüler:innen auf nahezu das gesamte Wissen des Abistoffes zurückgreifen können. Dieses Projekt basiert ebenso darauf, dass bereits alle vorangeschriebenen Lernbereiche behandelt wurden und die Schüler:innen hier vor den Abiturprüfungen noch einmal eine breite Auswahl der Themenfelder der Informatik wiederholen und anwendend verknüpfen können.

Ein besonderes Merkmal liegt hier jedoch auf Programmierkenntnissen in Python und Kenntnisse der objektorientierten Programmierung. Des weiteren werden vor allem die Lernbereiche „Datenstrukturen und Algorithmen“, „Softwareentwicklung“ und „Informationssicherheit“ angesprochen.

5. Kurzdarstellung

Empfohlene Python-Bibliotheken zum weiterreichen an die Schüler:innen:

Pandas

Pandas ist eine weit verbreitete Bibliothek zur Datenanalyse und -verarbeitung. Sie liefert praktische Datentypen wie das von mir verwendete pandas.DataFrame (ein zweidimensionales Array, auf welches wie auf Dictionarys zugreifbar ist und aus welchem sich direkt plotten lässt).

matplotlib.pyplot

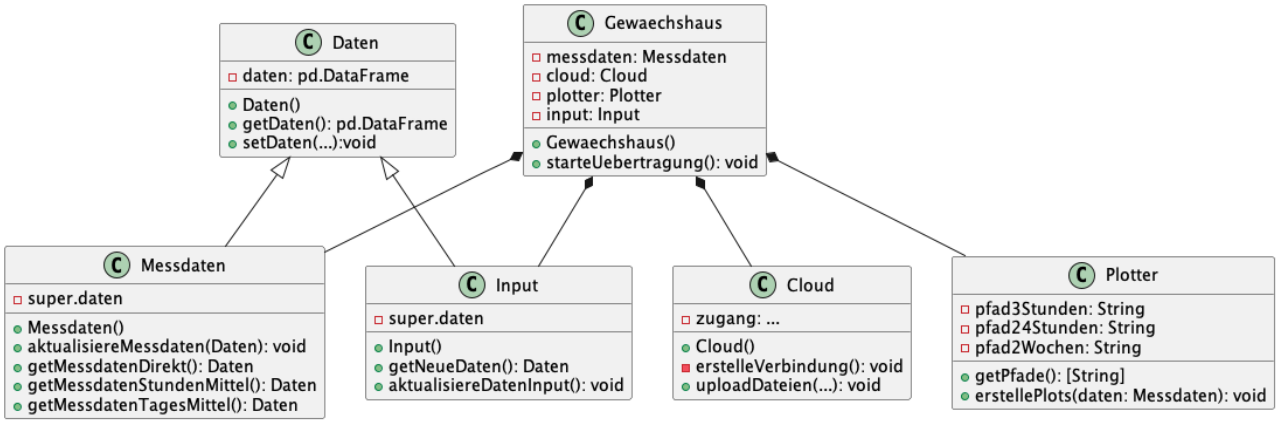
Eine Bibliothek zum Erstellen von Plots

Dropbox

Ermöglicht das Verbinden und Zugreifen auf den persönlichen Cloudspeicher des Anbieters Dropbox.

Bei Verwendung eines anderen Cloudanbieters sollte stattdessen eine andere entsprechende Bibliothek eingebunden werden.

Ab hier folgt eine Beispielvariante des Codes des automatischen Gewächshauses:



gewaechsaus.py:

```
import messdaten
import cloud
import input
import plotter

class Gewaechshaus:
   def __init__(self):
      pass
   def start(self):
      pass
```

```
class Gewaechshaus:
   def __init__(self):
      self.messdaten = messdaten.Messdaten()
      self.cloud = cloud.Cloud()
      self.plotter =plotter.Plotter()
      self.input = input.Input()
```

```
    def start(self):

        self.messdaten.simulliereDaten()

        print(self.messdaten.getMessdatenDirekt())

        while True:
            self.input.aktualisiereDatenInput()
            self.messdaten.aktualisiereMessdaten(self.input)
            self.plotter.erstellePlots(self.messdaten)
            print(self.messdaten.getMessdatenDirekt())
            self.cloud.uploadDatei(self.plotter.getPfade(),('/a_gewaechshaus3Stunden.png', '/
b_gewaechshaus1Tag.png', '/c_gewaechshaus2Wochen.png'))
            print('')

        print(self.messdaten.getMessdatenDirekt())



def main():
    print("Übertragung des smarten Gewächshauses wird gestartet")
    gwh = Gewaechshaus()
    gwh.start()
    print("Übertragung des automatischen Gewächshauses ist beendet")

if __name__ == "__main__":
    main()
```

daten.py:

```
from datetime import datetime
import pandas as pd

class Daten:
    def __init__(self):
```

```
        self.daten = pd.DataFrame({
            'LuftTemperatur': [],
            'BodenFeuchte': [],
            'LuftFeuchte': [],
            'Pumpenbenutzung': [],
            'Messzeitpunkt': []
        })
        self.daten.set_index('Messzeitpunkt', inplace=True)

    def getDaten(self):
        return self.daten.copy()

    def setDaten(self, feuchteBoden: [int], feuchteLuft: [int], luftTemperatur: [float], pumpenbenutzung:
[int], messzeitpunkt: [datetime]):
        self.daten['BodenFeuchte']=feuchteBoden
        self.daten['LuftFeuchte']=feuchteLuft
        self.daten['LuftTemperatur']=luftTemperatur
        self.daten['Pumpenbenutzung']=pumpenbenutzung
        self.daten['Messzeitpunkt']=messzeitpunkt

        self.daten.set_index('Messzeitpunkt', inplace=True)

    def setBodenFeuchte(self, bodenFeu: int):
        self.daten['BodenFeuchte'] = [bodenFeu]

    def setLuftTemperatur(self, luftTemp: float):
        self.daten['LuftTemperatur'] = [luftTemp]

    def setLuftFeuchte(self, LuftFeu: int):
        self.daten['LuftFeuchte'] = [LuftFeu]

    def setPumpe(self, pumpenbenutzung: int):
        self.daten['Pumpenbenutzung']=[pumpenbenutzung]

    def setMesszeitpunkt(self, messzeitpunkt: datetime):
```

```
            # self.daten['Messzeitpunkt']=[messzeitpunkt]
            self.daten.index = [messzeitpunkt]
```

messdaten.py:

```
from datetime import datetime, timedelta
import pandas as pd
import numpy as np

import daten

class Messdaten(daten.Daten):
    def __init__(self):
        super().__init__()

    def aktualisiereMessdaten(self, neueDaten: daten):
        self.daten = pd.concat([self.daten, neueDaten.getDaten()], sort=True)
        self.daten.sort_index(inplace=True)
        self.__entferneAlteDaten()

    def getMessdatenDirekt(self):
        return self.getDaten()

    def getMessdaten3Stunden(self):
        return self.daten[self.daten.index >= self.daten.index[-1] - timedelta(hours=3)]

    def getMitteldaten24Stunden(self):
        df = self.__getMitteldaten('h')
        return df[df.index >= self.daten.index[-1] - timedelta(hours=24)]

    def getMitteldaten2Wochen(self):
        df = self.__getMitteldaten('24h')
        return df[df.index >= self.daten.index[-1] - timedelta(weeks=2)]
```

```
def __getMitteldaten(self, mittlungszeit: str):
    df = self.getDaten()
    df = df.resample(mittlungszeit).agg({
        'LuftTemperatur': 'mean',
        'BodenFeuchte': 'mean',
        'LuftFeuchte': 'mean',
        'Pumpenbenutzung': 'sum'
    })
    return df


def __entferneAlteDaten(self):
    pass

def simulliereDaten(self):
    # 1. Erstellen eines Datetime-Indexes in 10-Minuten-Schritten
    start_time = '2024-06-15 00:00:00'
    end_time = '2024-07-01 00:00:00'
    time_index = pd.date_range(start=start_time, end=end_time, freq='10 min')

    # 2. Generieren der Temperaturwerte im Bereich von 17 bis 30 °C
    humidity_ground = np.random.uniform(55, 80, size=len(time_index))

    temperature_air = np.random.uniform(17, 28, size=len(time_index))

    # 3. Generieren der Feuchtewerte im Bereich von 35 bis 75 %
    humidity_air = np.random.uniform(35, 75, size=len(time_index))

    # 4. Generieren der Booleans für die Pumpenbenutzung
    pump_usage = np.random.choice([0, 1], size=len(time_index))

    # 5. Erstellen des DataFrames
    self.setDaten(humidity_ground, humidity_air, temperature_air, pump_usage, time_index)
```

input.py:

```
from datetime import datetime
import serial
import time

import daten

class Input(daten.Daten):
    def __init__(self):
        super().__init__()
        self.a = serial.Serial('/dev/ttyACM0', 9600, timeout=1.0)

    def getNeueDaten(self):
        return self.daten.copy()

    def aktualisiereDatenInput(self):

        while (not(self.a.inWaiting()>0)):
            time.sleep(10)
        temp = self.a.readline().decode('ascii')
        feuchteLuft = self.a.readline().decode('ascii')
        feuchteBoden = self.a.readline().decode('ascii')
        pumpe = self.a.readline().decode('ascii')

        self.setDaten(int(feuchteBoden), int(feuchteLuft), float(temp), int(pumpe), datetime.now())
```

plotter.py:

```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import pandas as pd
import numpy as np
import datetime
```

```
import messdaten

class Plotter:
    def __init__(self):
        self.pfad3Stunden = '../plots/plot3Stunden.png'
        self.pfad24Stunden = '../plots/plot24Stunden.png'
        self.pfad2Wochen = '../plots/plot2Wochen.png'

    def getPfade(self):
        return (self.pfad3Stunden, self.pfad24Stunden, self.pfad2Wochen)

    def erstellePlots(self, messdaten):
        self.__erstellePlot3Stunden(messdaten)
        self.__erstellePlot24Stunden(messdaten)
        self.__erstellePlot2Wochen(messdaten)

    def __erstellePlot3Stunden(self, messdaten):

        df = messdaten.getMessdaten3Stunden()

        fig = plt.figure(figsize=(8, 9))
        gs = GridSpec(3, 1, height_ratios=[3, 3, 2])  # Letzter Plot kleiner machen

        # Plot für Temperatur
        ax0 = fig.add_subplot(gs[0])
        ax0.plot(df.index, df['LuftTemperatur'], 'r.-', label='LuftTemperatur')
        ax0.set_ylabel('Temperatur')

        # Plot für Feuchte
        ax1 = fig.add_subplot(gs[1], sharex=ax0)
        ax1.plot(df.index, df['BodenFeuchte'], 'c^-', label='BodenFeuchte')
        ax1.plot(df.index, df['LuftFeuchte'], 'b.-', label='LuftFeuchte')
        ax1.set_ylabel('relative Feuchte')
        ax1.legend()
```

```
        # Plot für Pumpe
        ax2 = fig.add_subplot(gs[2], sharex=ax0)
        ax2.plot(df.index, df['Pumpenbenutzung'], 'ko-')
        ax2.set_ylim(-0.4, 1.4)
        ax2.set_ylabel('Pumpe')
        ax2.set_yticks([0, 1])
        ax2.set_yticklabels(['Aus', 'An'])
        ax2.set_xlabel('Messzeitpunkt')

        # Gemeinsamen Titel hinzufügen
        fig.suptitle('Messdaten der letzten 3 Stunden', fontsize=16)

        # Layout anpassen
        plt.tight_layout(rect=[0, 0, 1, 0.96])

        # Speichern des Plots
        plt.savefig(self.pfad3Stunden)
        plt.close(fig)

    def __erstellePlot24Stunden(self, messdaten):
        daten = messdaten.getMitteldaten24Stunden()
        self.__erstellePlotgemittelt(daten, 'stündlich gemittelte Messdaten der letzten 24 Stunden',
self.pfad24Stunden)

    def __erstellePlot2Wochen(self, messdaten):
        daten = messdaten.getMitteldaten2Wochen()
        self.__erstellePlotgemittelt(daten, 'täglich gemittelte Messdaten der letzten 2 Wochen',
self.pfad2Wochen)

    def __erstellePlotgemittelt(self, daten, title: str , pfad):

        fig = plt.figure(figsize=(8, 9))
        gs = GridSpec(3, 1, height_ratios=[3, 3, 2])  # Letzter Plot kleiner machen
```

```
    # Plot für Temperatur
    ax0 = fig.add_subplot(gs[0])
    ax0.plot(daten.index, daten['LuftTemperatur'], 'r.-')
    ax0.set_ylabel('mittlere Temperatur')

    # Plot für Feuchte
    ax1 = fig.add_subplot(gs[1], sharex=ax0)
    ax1.plot(daten.index, daten['BodenFeuchte'], 'c^-', label='Bodenfeuchte')
    ax1.plot(daten.index, daten['LuftFeuchte'], 'b.-', label='Luftfeuchte')
    ax1.set_ylabel('mittlere Feuchte')
    ax1.legend()

    # Plot für Pumpe
    ax2 = fig.add_subplot(gs[2], sharex=ax0)
    ax2.plot(daten.index, daten['Pumpenbenutzung'], 'ko-')
    ax2.set_ylabel('Anzahl der Pumpennutzungen')
    ax2.set_xlabel('Messzeitpunkt')

    # Gemeinsamen Titel hinzufügen
    fig.suptitle(title, fontsize=16)

    # Layout anpassen
    plt.tight_layout(rect=[0, 0, 1, 0.96])

    # Speichern des Plots
    plt.savefig(pfad)
    plt.close(fig)
```

cloud.py:

```
import dropbox
from dropbox import DropboxOAuth2FlowNoRedirect
```

```
class Cloud:
    def __init__(self):
        self.zugang = None
        self.__erstelleVerbindung()

    def __erstelleVerbindung(self):
        APP_KEY = "kkv6pga1dgvmusu"

        auth_flow = DropboxOAuth2FlowNoRedirect(APP_KEY, use_pkce=True, token_access_type='offline')

        authorize_url = auth_flow.start()
        print("1. Go to: " + authorize_url)
        print("2. Click \"Allow\" (you might have to log in first).")
        print("3. Copy the authorization code.")
        auth_code = input("Enter the authorization code here: ").strip()

        try:
            oauth_result = auth_flow.finish(auth_code)
        except Exception as e:
            print('Error: %s' % (e,))
            exit(1)

        self.zugang = dropbox.Dropbox(oauth2_refresh_token=oauth_result.refresh_token,
app_key=APP_KEY)

    def uploadDatei(self, dateipfad, zielpfad):

        for i in range(3):
            with open(dateipfad[i], 'rb') as f:
                try:
                    self.zugang.files_upload(f.read(), zielpfad[i], mode=dropbox.files.WriteMode.overwrite)
                    print(f"File {dateipfad[i]} uploaded to {zielpfad[i]}.")
                except Exception as e:
                    print(f"Error uploading file: {e}")
```