

Lösungsblatt

Diese Fassung enthält die Lösungen.

Wie die Aufgaben aufgebaut sind

Standard ist das Ziel, fang immer dort an. **Einstieg** führt hin, falls du nicht reinkommst.

Experte geht weiter, wenn du fertig bist. Nicht jede Aufgabe hat alle drei Stufen.

Zum Nachschlagen liegen **Kärtchen** in der Tischmitte.

Heute läuft alles auf dem Laptop

Stell Thonny unten rechts auf "Lokales Python 3" um. Solange dort "MicroPython (ESP32)" steht, findet dein Programm die Bibliotheken von heute nicht.

Umgekehrt gilt genauso: machine, Pin und I2C gibt es hier nicht. Ein Programm von gestern endet im ImportError.

Die drei Aufgaben ergeben ein Programm

Aufgabe 2 hängt an das Programm aus Aufgabe 1 an, Aufgabe 3 an das aus Aufgabe 2.

Speichere deine Datei, statt jedes Mal neu anzufangen.

Aufgabe 1 – Die eigenen Messwerte holen

Standard

a) Die Umgebung prüfen

Bevor du anfängst, zwei Zeilen zur Kontrolle. Tipp sie in die Shell, nicht in den Editor:

```
import requests
requests.__version__
```

Kommt eine Versionsnummer, ist alles bereit. Kommt ein ImportError, stimmt etwas mit der Umgebung nicht. Was könnte das sein?

Lösung

Zwei mögliche Ursachen, und die erste ist die häufigere:

- Thonny steht noch auf "MicroPython (ESP32)". Dann sucht die Zeile die Bibliothek auf dem Board, und dort liegt sie nicht.
- Thonny steht richtig, aber requests ist nicht installiert. Nachinstallieren über *Werkzeuge, Pakete verwalten*.

requests gehört nicht zum Grundumfang von Python. Sie ist die meistbenutzte Bibliothek, um Adressen im Netz aufzurufen, aber eben doch eine Bibliothek wie bme280 an Tag 3.

b) Die Anfrage bauen

Vier Schnipsel, einer nach dem anderen in den Editor. Zusammen ergeben sie dein Programm.

Erstens: die Bibliothek und die zwei Angaben, die dir gehören.

```
import requests

TOKEN = "hier steht dein Token"
ADRESSE = "https://www.loramint.de/api/v1/measurements"
```

Den Token bekommst du ausgeteilt. Er ist dein Ausweis gegenüber dem Server und erlaubt nur das Lesen, du kannst damit nichts kaputt machen.

Zweitens: die Anfrage abschicken.

```
antwort = requests.get(
    ADRESSE,
    headers={"Authorization": "Bearer " + TOKEN},
    params={"per_page": 5},
)
```

requests.get holt etwas von einer Adresse. In headers steht der Ausweis, in params stehen die Wünsche: hier fünf Messwerte statt der üblichen 20. Beides sind Dictionaries, genau wie an Tag 3.

Warum so wenige? Weil du gleich nachliest, was zurückkommt. Fünf Einträge passen auf einen Bildschirm, hundert nicht.

Drittens: nachsehen, ob es geklappt hat.

```
print(antwort.status_code)
```

Viertens: den Text in Python-Daten verwandeln und ausgeben lassen.

```
daten = antwort.json()
print(daten)
```

Über die Leitung kommt Text. .json() macht daraus ein Dictionary, mit dem du arbeiten kannst.

Fünftens: die Ausgabe in Ruhe ansehen. Kopier sie in eine leere Datei und überleg dir, welche Teile davon was bedeuten könnten.

Die Antwort hat zwei Teile. Wie heißen sie, und was steht in jedem drin?

Lösung

```
{"data": [ ... ], "pagination": { ... }}
```

data die eigentlichen Messwerte, als Liste. Jeder Eintrag darin ist wieder ein Dictionary.

pagination keine Messwerte, sondern Angaben über die Antwort selbst: wie viele es insgesamt sind und auf wie viele Seiten sie verteilt wurden.

Die Ausgabe ist unübersichtlich, und das ist normal. Über die Leitung kommt alles in einer Zeile, ohne Zeilenumbrüche und ohne Einrückung. Erst wer die geschweiften und eckigen Klammern paarweise zuordnet, sieht die Struktur. Genau dafür wird die Ausgabe in eine Datei kopiert.

Wer es bequemer will: `import json` und dann `print(json.dumps(daten, indent=2))` rückt die Ausgabe ein. Das ist kein Pflichtstoff, hilft beim Lesen aber sofort.

c) Nachsehen, was angekommen ist

Häng zwei Zeilen an und starte das Programm.

```
print(len(daten["data"]), "Messwerte")
print(daten["data"][0])
```

Trag ein, was du siehst:

Frage	Deine Antwort
Welche Zahl steht als Statuscode?	
Wie viele Messwerte sind gekommen?	
Welche Felder hat ein einzelner Eintrag?	

Lösung

```
# Die eigenen Messwerte holen.

import requests

TOKEN = "hier steht dein Token"
ADRESSE = "https://www.loramint.de/api/v1/measurements"

antwort = requests.get(
    ADRESSE,
    headers={"Authorization": "Bearer " + TOKEN},
    params={"per_page": 5},
)

print(antwort.status_code)

daten = antwort.json()
print(daten)

print(len(daten["data"]), "Messwerte")
print(daten["data"][0])
```

Statuscode 200. Anzahl genau 5, mehr wurden nicht angefragt. Wie viele es insgesamt gibt, steht in pagination unter total.

	value	der Messwert, als Text
	unit	die Einheit
	measurand	die Messgröße
Die Felder eines Eintrags:	sensor	der Sensor
	location	der Ort
	deviceEui	welche Node gesendet hat
	recordedAt	wann gemessen wurde

Das sind genau die fünf Angaben aus `MintValue` von gestern, plus Node und Zeitpunkt. Was oben ins Funkmodul ging, kommt hier unten wieder heraus.

Experte

Bisher bekommst du alles, was dein Token sehen darf. Schränk die Anfrage auf das ein, was dich interessiert.

Filter	Was er tut
device_eui	nur eine bestimmte Node, 16 Zeichen
measurand	nur eine Messgröße, etwa "Temperatur"
location	nur einen Ort
from, to	nur einen Zeitraum

Die Filter kommen zu `per_page` in `params` dazu. Die EUI deiner Node steht auf dem Protokollbogen von gestern.

Die API beschreibt sich selbst

Unter `loramint.de/api/v1/docs` steht jede Adresse mit allen Angaben, die sie annimmt, und mit dem, was sie zurückgibt. Zu jeder lässt sich dort auch gleich eine Anfrage ausprobieren.

Das ist keine Anleitung für Anfänger, sondern eine Nachschlagestelle. Genau damit arbeiten Leute, die eine fremde API benutzen.

Beantworte mit Hilfe der Dokumentation:

1. In welchem Format erwarten from und to den Zeitpunkt?
2. Welche Filter gibt es außer den vier oben?

Sieh dir außerdem `daten["pagination"]` an und beantworte: Wie viele Messwerte gibt es insgesamt, und hast du sie alle bekommen?

Lösung

```
antwort = requests.get(
    ADRESSE,
    headers={"Authorization": "Bearer " + TOKEN},
    params={
        "device_eui": "a8404152825f8d0f",
        "measurand": "Temperatur",
        "per_page": 100,
    },
)

print(daten["pagination"])
```

In pagination stehen total und total_pages. Ist total_pages größer als 1, fehlt dir etwas: Dann liegt der Rest auf der nächsten Seite, und du brauchst "page": 2.

100 ist das Maximum. Ein größeres per_page nimmt die API nicht an, sie antwortet mit 400. Wer mehr will, muss blättern. Das ist Absicht: Ein Server, der auf eine einzige Anfrage hin beliebig viel herausgibt, lässt sich mit einer Zeile lahmlegen.

Achtung, eine echte Stolperstelle. Der Filter heißt device_eui mit Unterstrich, das Feld in der Antwort aber deviceEui ohne. Das ist kein Tippfehler auf diesem Blatt, das steht in der API so. Wer den Filter falsch schreibt, bekommt keine Fehlermeldung, sondern alle Messwerte statt einer Node.

Aus der Dokumentation:

Zeitformat	entweder nur der Tag, "2026-09-10", oder Tag und Uhrzeit mit Zeitzone, "2026-09-10T20:00:00+02:00"
weitere Filter	sensor, datatype, group_name, public_read

Für die Maschinenfassung gibt es `/api/v1/openapi.json`, das ist dieselbe Beschreibung ohne Oberfläche.

S051

S056

Aufgabe 2 – Rechnen mit echten Zahlen

Standard

a) Ein einzelner Messwert, genau angesehen

Häng die Zeilen an dein Programm aus Aufgabe 1 an und starte es.

```
# Ein einzelner Messwert, genau angesehen.  
# Diese Zeilen kommen ans Ende deines Programms aus Aufgabe 1.  
  
messwerte = daten["data"]  
messwert = messwerte[0]  
  
print("Der ganze Messwert:", messwert)  
print("Nur der Wert:      ", messwert["value"])  
print("Sein Datentyp:     ", type(messwert["value"]))  
print("Plus eins:         ", messwert["value"] + 1)
```

Die letzte Zeile führt zu einem Fehler. Was steht in der Meldung, und warum passiert das?

Lösung

```
TypeError: can only concatenate str (not "int") to str
```

`messwert["value"]` ist eine **Zeichenkette**, keine Zahl. Die Ausgabe verrät es zweimal: an den Anführungszeichen um "13.4" und an `<class 'str'>` beim Datentyp.

Genau wie an Tag 3. Dort lieferte `sensor_bme.values` Text mit Einheit, und ihr seid auf `read_compensated_data` ausgewichen. Hier gibt es kein Ausweichen: `float()` macht aus dem Text eine Zahl.

b) Aus Text werden Zahlen

Drei Schritte, bis die Zahlen beieinanderliegen.

Erstens: mehr Daten holen. Für einen Mittelwert über die Nacht reichen fünf Werte nicht. Setz `per_page` auf **100**, mehr gibt die API auf einmal nicht her.

Sieh danach in `daten["pagination"]` nach, wie viele es insgesamt sind. Steht dort mehr als 100, fehlt dir ein Teil der Nacht. Halt an der Stelle kurz inne: Über welchen Zeitraum rechnest du dann eigentlich?

Zweitens: die Reihenfolge umdrehen. Die API antwortet mit dem **neuesten Wert zuerst**. Für den Mittelwert ist das gleichgültig, für das Diagramm in Aufgabe 3 nicht: Dort lief die Zeit sonst rückwärts. Dreh die Liste gleich hier um, dann stimmt sie für den Rest des Tages.

```
messwerte = daten["data"]  
messwerte.reverse()
```

Drittens: eine leere Liste anlegen. Sie nimmt gleich die Zahlen auf.

```
werte = []
```

Viertens: die Liste füllen. Geh mit einer Schleife durch alle Messwerte und häng jeden einzelnen an. Was in die Lücke gehört, weißt du aus Teil a).

```
for messwert in messwerte:  
    werte.append(_____)
```

Lass zur Kontrolle `print(werte)` laufen. Stehen dort Zahlen ohne Anführungszeichen, hat es geklappt.

Lösung

```
messwerte = daten["data"]
messwerte.reverse()

werte = []
for messwert in messwerte:
    werte.append(float(messwert["value"]))
```

In die Lücke gehört `float(messwert["value"])`. Die Schleife geht durch die Messwerte, holt aus jedem das Feld `value`, macht eine Zahl daraus und hängt sie an `werte` an.

Zwei Listen, zwei Aufgaben. `messwerte` enthält die Dictionaries, so wie sie vom Server kamen. `werte` enthält nur die Zahlen. Wer beides verwechselt, bekommt genau den Fehler aus Teil a).

Zur Frage nach dem Zeitraum. Die Antwort kommt neueste zuerst. Wer 100 von 500 Werten holt, rechnet über die letzten Stunden der Nacht und nicht über die ganze. Wer alles will, blättert:

```
messwerte = []
seite = 1
while True:
    antwort = requests.get(ADRESSE, headers=kopf,
                           params={"per_page": 100, "page": seite})
    daten = antwort.json()
    messwerte = messwerte + daten["data"]
    if not daten["pagination"]["has_next"]:
        break
    seite = seite + 1
```

Das ist die `while`-Schleife aus dem Gruppenpuzzle von Tag 2, mit `break` aus Aufgabe 6 von Arbeitsblatt 2. Für die Pflichtaufgabe ist sie nicht nötig; wer von selbst darauf kommt, hat den Tag verstanden.

Ohne `float()` läuft die Schleife durch, und erst das Rechnen in Teil c) bricht ab. Der Fehler zeigt dann auf die falsche Zeile. Genau deshalb steht die Umwandlung dort, wo der Wert entsteht.

c) Rechnen

Jetzt ist `werte` eine gewöhnliche Liste von Zahlen, und alles Weitere kennst du. Gesucht sind drei Werte:

Gesucht	Womit
Mittelwert	<code>sum()</code> und <code>len()</code> , gerundet auf eine Nachkommastelle
Höchstwert	<code>max()</code>
Tiefstwert	<code>min()</code>

Die drei Funktionen kennst du aus Aufgabe 6 von Arbeitsblatt 1. Gib zu jedem Ergebnis dazu, was es ist, sonst stehen am Ende drei nackte Zahlen da.

Lösung

```
# Aus den Messwerten eine Liste von Zahlen machen und damit rechnen.

messwerte = daten["data"]
messwerte.reverse()      # die API antwortet neueste zuerst

werte = []
for messwert in messwerte:
    werte.append(float(messwert["value"]))

print("Anzahl: ", len(werte))
print("Mittel: ", round(sum(werte) / len(werte), 1))
print("Höchster:", max(werte))
print("Tiefster:", min(werte))
```

$\text{sum(werte)} / \text{len(werte)}$ ist der Mittelwert, $\text{round}(\dots, 1)$ schneidet ihn auf eine Nachkommastelle. Ohne das Runden stehen dort fünfzehn Stellen, von denen keine einzige gemessen wurde.

Der Mittelwert einer Nacht ist mit Vorsicht zu lesen. Er verschweigt, dass es am Anfang wärmer war als am Ende. Genau deshalb kommt in Aufgabe 3 das Diagramm dazu.

Experte

Der Höchstwert allein sagt wenig. Interessant wird er mit dem Zeitpunkt.

Gib zum höchsten und zum tiefsten Wert aus, **wann** er gemessen wurde. Der Zeitpunkt steht im Feld `recordedAt`.

Lösung

Ein Weg über die Messwerte selbst, statt über die Liste der Zahlen:

```
waermster = messwerte[0]
for messwert in messwerte:
    if float(messwert["value"]) > float(waermster["value"]):
        waermster = messwert

print("Höchster Wert:", waermster["value"], "um", waermster["recordedAt"])
```

Das ist der wärmste Tag aus Aufgabe 5 von Arbeitsblatt 2, nur mit echten Daten. Kürzer geht es mit $\text{max(messwerte, key=...)}$, aber die Schleife zeigt besser, was passiert.

S052

S053

Aufgabe 3 – Aus Zahlen wird ein Bild

Standard

Drei Schnipsel, und deine Nacht steht als Kurve auf dem Bildschirm.

Läuft deine Kurve verkehrt herum?

Dann fehlt das `messwerte.reverse()` aus Aufgabe 2. Die API antwortet mit dem neuesten Wert zuerst, und ohne das Umdrehen fängt dein Diagramm am Morgen an.

Erstens: die Bibliothek.

```
import matplotlib.pyplot as plt
```

`as plt` gibt der Bibliothek einen kurzen Namen. Sonst stünde `matplotlib.pyplot` vor jeder einzelnen Zeile.

Zweitens: zeichnen und zeigen.

```
plt.plot(werte)
plt.show()
```

Drittens: beschriften. Ergänze Titel und beide Achsen, *bevor* du `show()` aufrufst. Danach ist das Fenster schon da.

Funktion	Was sie beschriftet
<code>plt.title()</code>	die Überschrift des Diagramms
<code>plt.xlabel()</code>	die waagerechte Achse
<code>plt.ylabel()</code>	die senkrechte Achse, hier mit der Einheit

Zum Schluss speichern mit `plt.savefig("nacht.png")`. Auch diese Zeile gehört vor `show()`.

Lösung

```
# Aus Zahlen wird ein Bild.

import matplotlib.pyplot as plt

plt.plot(werte)

plt.title("Temperatur über Nacht")
plt.xlabel("Messung")
plt.ylabel("Grad Celsius")

plt.savefig("nacht.png")
plt.show()
```

Die Reihenfolge ist der Stolperstein. `show()` übergibt das Fenster an den Benutzer und leert danach die Zeichnung. Alles, was beschriftet oder gespeichert werden soll, muss vorher passieren. Wer hinterher speichert, bekommt ein leeres Bild und keine Fehlermeldung.

Ohne Achsenbeschriftung behauptet ein Diagramm etwas, das es nicht zeigt. Eine Kurve, die von 20 auf 13 fällt, sagt nicht, ob es um Grad, Prozent oder Stunden geht. Das ist derselbe Gedanke wie beim Dictionary an Tag 3: Eine Zahl allein sagt nichts.

Experte

Fünf Ausbaustufen. Such dir aus, was dich interessiert, und arbeite so weit, wie du kommst. Das ist freiwillig.

Ausbaustufe

- a Eine zweite Messgröße dazu, mit eigener Achse
- b Nacht und Tag im Diagramm farbig hinterlegen
- c Höchst- und Tiefstwert im Diagramm beschriften
- d Zwei Orte übereinanderlegen und vergleichen
- e Das Bild als Datei speichern, mit Titel und Beschriftung

Lösung

Ansatzpunkte, keine fertigen Programme. Die Teilnehmenden sollen suchen.

- a `plt.gca().twinx()` gibt eine zweite senkrechte Achse, dazu eine zweite Anfrage mit anderem `measurand`
- b `plt.axvspan()` hinterlegt einen Bereich farbig
- c `plt.annotate()`, oder ein roter Punkt über `plt.plot(x, y, "ro")`
- d zwei Anfragen mit verschiedenem `location`, zweimal `plt.plot()`, dazu `plt.legend()`
- e `plt.savefig()` VOR `show()`

Stufe d vorher prüfen: Wenn alle Nodes im selben Raum standen, ist sie nur eine Enttäuschung.

S054

S055

Anhang • Messwerte abrufen und auswerten

Alles von den Kärtchen noch einmal, zum Mitnehmen und Nachschlagen.

Eine Anfrage stellen

```
import requests

antwort = requests.get(
    "https://www.loramint.de/api/v1/measurements",
    headers={"Authorization": "Bearer " + TOKEN},
    params={"measurand": "Temperatur", "per_page": 100},
)

daten = antwort.json()
```

Teil	Was er tut
die Adresse	wohin die Anfrage geht
headers	der Ausweis, der Token gehört hierher und nicht in die Adresse
params	die Wünsche: Filter und Seitengröße
.json()	macht aus dem Text ein Dictionary

Filter und Seiten

Angabe	Wirkung
device_eui	nur eine Node, 16 Zeichen
measurand	nur eine Messgröße
sensor	nur einen Sensor
location	nur einen Ort
from, to	nur einen Zeitraum
per_page	wie viele Werte je Seite, Vorgabe 20, höchstens 100
page	welche Seite

In `daten["pagination"]` steht, wie viele es insgesamt sind.

Was in der Antwort steht

```
{
  "data": [
    {
      "measurand": "Temperatur",
      "value": "13.4",
      "unit": "°C",
      "location": "Raum 101"
    }
  ],
  "pagination": {
    "page": 1,
    "per_page": 20,
    "total": 240
  }
}
```

Feld	Inhalt
value	der Messwert, als Text
unit	die Einheit
measurand	die Messgröße
sensor	der Sensor
location	der Ort
deviceEui	welche Node gesendet hat
recordedAt	wann gemessen wurde

Die Filter heißen anders als die Felder. Beim Fragen `device_eui`, beim Lesen `deviceEui`.

Aus Text eine Zahl machen

```
messwerte = daten["data"]
messwerte.reverse()

werte = []
for messwert in messwerte:
    werte.append(float(messwert["value"]))

print(round(sum(werte) / len(werte), 1))
print(max(werte), min(werte))
```

Die Antwort kommt neueste zuerst, deshalb das `reverse()`. Ohne es läuft die Zeit im Diagramm rückwärts.

`float()` an der Stelle, wo der Wert entsteht. Dann ist der Rest eine gewöhnliche Liste von Zahlen.

Ein Diagramm zeichnen

```
import matplotlib.pyplot as plt

plt.plot(werte)
plt.title("Temperatur über Nacht")
plt.xlabel("Messung")
plt.ylabel("Grad Celsius")
plt.savefig("nacht.png")
plt.show()
```

`show()` **kommt zuletzt**. Beschriften und Speichern vorher, sonst ist die Zeichnung schon geleert.

Wenn etwas schiefgeht

Was passiert	Was zu tun ist
ImportError	Thonny steht noch auf MicroPython, oder die Bibliothek fehlt.
Statuscode 401	Der Token stimmt nicht. Auf Leerzeichen am Anfang und Ende achten.
Statuscode 404	Die Adresse stimmt nicht. <code>/api/v1/measurements</code> nachlesen.
Statuscode 200, aber data ist leer	Der Filter passt auf nichts. Messgröße und Ort genau so schreiben wie beim Senden.
TypeError beim Rechnen	<code>float()</code> vergessen. <code>value</code> ist Text.
KeyError	Ein Feldname ist falsch geschrieben. Groß und klein zählt.
Das Bild ist leer	Gespeichert wurde nach <code>show()</code> .

Zwei Minuten suchen, dann Hand heben.