

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 – nimm dir, was du brauchst, und leg es zurück, wenn du fertig bist.

Aufgabe 1 – Eine Messreihe in einer Liste

Standard

Öffne `messreihe.py` – die Woche steht schon drin:

```
# Eine Woche Temperaturen aus dem Schulgarten, jeden Mittag gemessen.  
# Sieben Werte, ein Name.  
  
messwerte = [12.4, 15.1, 18.9, 21.8, 19.3, 14.7, 16.2]  
  
print("Erster Wert:", messwerte[0])
```

Ergänze das Programm, sodass es außerdem ausgibt:

- a) den letzten Wert
- b) wie viele Werte es sind
- c) den wärmsten und den kältesten Messwert
- d) den Mittelwert, auf eine Nachkommastelle gerundet

Lösung

```
print("Letzter Wert:", messwerte[6])  
print("Wie viele:", len(messwerte))  
print("Waermster:", max(messwerte))  
print("Kaeltester:", min(messwerte))  
print("Im Mittel:", round(sum(messwerte) / len(messwerte), 1))
```

`messwerte[6]` ist der siebte Wert – gezählt wird ab null.

Experte

Am Sonntag wird noch einmal gemessen: 17.5. Häng den Wert an die Liste an, ohne sie neu zu schreiben:

```
messwerte.append(17.5)
```

Starte das Programm erneut. Vier deiner fünf Ausgaben stimmen sofort, eine nicht. Welche, und warum?

Lösung

Falsch ist nur `messwerte[6]` – der letzte Wert ist jetzt der achte. `len`, `max`, `min` und `sum` rechnen dagegen mit der ganzen Liste, egal wie lang sie ist.

Sicherer: `messwerte[-1]` nimmt immer den letzten Wert.

S021

Aufgabe 2 – Eine Warnung ausgeben

Einstieg

Öffne `warnung.py`.

```
# Eine einzelne Messung wird geprüeft.  
# Ändere den Wert und starte erneut.  
  
temperatur = 18.0  
  
if temperatur > 20:  
    print("Zu warm!")  
  
print("Fertig.")
```

Sag mit dem Kärtchen **Verzweigen** voraus, was herauskommt. Probier es danach aus.

temperatur	Deine Vorhersage	Tatsächlich
18.0		
21.5		
20.0		

Lösung

temperatur	Ausgabe	Warum
18.0	nur Fertig.	18.0 ist nicht größer als 20
21.5	Zu warm! und Fertig.	Bedingung trifft zu
20.0	nur Fertig.	20 ist nicht <i>größer</i> als 20

Fertig. kommt immer: Die Zeile ist nicht eingerückt und gehört damit nicht zur Verzweigung.

Standard

Erweitere `warnung.py` so, dass es bei jedem Wert etwas sagt – bei zu warmen Werten eine Warnung, sonst eine Entwarnung.

Lösung

```
if temperatur > 20:  
    print("Zu warm!")  
else:  
    print("Alles im grünen Bereich.")
```

Der `else`-Zweig braucht keine Bedingung; er gilt für alles andere.

Experte

Statt etwas auszugeben, soll die LED die Antwort zeigen: rot bei über 20 Grad, grün darunter. Die LED-Zeilen stehen auf dem Kärtchen **Wiederholen** nicht – schau in `farben.py` oder auf dein Blatt von gestern.

Lösung

```
import machine, neopixel

led = neopixel.NeoPixel(machine.Pin(48), 1)

if temperatur > 20:
    led[0] = (30, 0, 0)
else:
    led[0] = (0, 30, 0)

led.write()
```

`led.write()` steht nur einmal und nicht in beiden Zweigen: Es schickt die Farbe zur LED, egal welche vorher gesetzt wurde.

S022

Aufgabe 3 – Immer wieder messen

Einstieg

Öffne `zaehlen.py` und starte es.

```
# Eine Schleife, die mitzaehlt.  
  
zaehler = 1  
  
while zaehler <= 5:  
    print("Messung Nummer", zaehler)  
    zaehler = zaehler + 1  
  
print("Fertig.")
```

- a) Wie viele Zeilen erscheinen?
- b) Ändere die 5 in eine 3. Sag vorher, was herauskommt.
- c) Ändere den Startwert von 1 auf 3. Wie viele Zeilen sind es jetzt?

Lösung

a) fünf. b) drei. c) drei – gezählt wird von 3 bis 5.

Die Anzahl ergibt sich aus Startwert und Bedingung zusammen, nicht aus der Zahl in der Bedingung allein.

Standard

Gib die ganze Messreihe aus Aufgabe 1 aus – mit Nummer davor, und zwischen zwei Werten eine halbe Sekunde Pause, als würde gerade gemessen:

```
Messung 1 : 12.4  
Messung 2 : 15.1
```

Die Zählvariable ist zugleich der Index: `messwerte[i]`. Für die Pause hilft das Kärtchen **Warten**.

Lösung

```
import time  
  
messwerte = [12.4, 15.1, 18.9, 21.8, 19.3, 14.7, 16.2]  
  
i = 0  
while i < len(messwerte):  
    print("Messung", i + 1, ":", messwerte[i])  
    time.sleep(0.5)  
    i = i + 1
```

`i < len(messwerte)` statt `i <= 6`: So stimmt die Schleife auch dann noch, wenn die Liste länger wird.

Experte

Zähle beim Durchlaufen mit, wie viele Werte über 20 Grad liegen.

Du brauchst einen **zweiten Zähler** – und eine `if`-Abfrage mit einem Vergleich, die ihn nur dann erhöht, wenn der Wert über 20 liegt. Die Kärtchen **Verzweigen** und **Wiederholen** zusammen.

Lösung

```
i = 0
zu_warm = 0

while i < len(messwerte):
    if messwerte[i] > 20:
        zu_warm = zu_warm + 1
    i = i + 1

print("Über 20 Grad:", zu_warm, "von", len(messwerte))
```

Zwei Zähler, die verschiedene Dinge zählen: `i` die Durchläufe, `zu_warm` die Treffer. Der zweite wird nur im `if` erhöht.

S023

S025

Aufgabe 4 – Blinkmuster

Einstieg

Öffne `farben.py` und starte es.

```
# Eine Liste von Farben, nacheinander auf der LED.  
  
import machine, neopixel, time  
  
led = neopixel.NeoPixel(machine.Pin(48), 1)  
  
farben = [(30, 0, 0), (0, 30, 0), (0, 0, 30)]  
  
for farbe in farben:  
    led[0] = farbe  
    led.write()  
    time.sleep(0.5)
```

- a) Wie oft läuft die Schleife? Woher weiß sie das?
- b) Häng eine vierte Farbe an die Liste. Was musstest du an der Schleife ändern?
- c) Ändere die `0.5`. Was passiert?

Lösung

a) dreimal, einmal je Eintrag der Liste. b) nichts. c) das Blinken wird schneller oder langsamer.

Das ist der Unterschied zu `while`: Bei `for` bestimmt die Liste, wie oft – man muss nichts mitzählen.

Standard

Bau dir ein eigenes Blinkmuster: mindestens fünf Farben, und am Ende soll die LED ausgehen.

Aus ist `(0, 0, 0)`.

Lösung

```
farben = [(30, 0, 0), (30, 30, 0), (0, 30, 0),  
          (0, 30, 30), (0, 0, 30)]  
  
for farbe in farben:  
    led[0] = farbe  
    led.write()  
    time.sleep(0.3)  
  
led[0] = (0, 0, 0)  
led.write()
```

Das Ausschalten steht **nach** der Schleife, nicht darin – sonst wäre die LED nach jeder Farbe kurz aus.

Experte

Jetzt alles zusammen: Lauf mit `for` über die Messreihe und lass die LED für jeden Wert kurz leuchten – rot, wenn er über 20 Grad liegt, sonst grün.

Lösung

```
for wert in messwerte:
    if wert > 20:
        led[0] = (30, 0, 0)
    else:
        led[0] = (0, 30, 0)
    led.write()
    time.sleep(0.5)
```

Die Verzweigung steht **in** der Schleife: Sie wird für jeden Wert neu entschieden. Damit blinkt die ganze Woche in zwei Farben durch – ein Messprotokoll, das man ansehen kann.

S024

S025

Aufgabe 5 – Der wärmste Tag ohne `max`

Standard

In Aufgabe 1 hat `max` den wärmsten Wert gefunden. Jetzt machst du es selbst: Bestimme ihn mit einer Schleife, **ohne** `max` zu benutzen.

Merk dir einen "bisher wärmsten" Wert und vergleiche jeden neuen damit. Ist der neue größer, wird er der neue gemerkte Wert.

Lösung

```
waermster = messwerte[0]

for wert in messwerte:
    if wert > waermster:
        waermster = wert

print("Waermster:", waermster)
```

Der gemerkte Wert startet auf `messwerte[0]` und nicht auf `0`: Bei einer Frostwoche wären sonst alle Werte kleiner als der Startwert, und das Ergebnis wäre `0` – ein Tag, den es nie gab.

Experte

Gib zusätzlich aus, an welchem Tag der wärmste Wert gemessen wurde – "Waermster: 21.8 an Tag 4".

Der Tag ist die Nummer des Werts in der Liste. Dafür brauchst du den Index, also wieder eine `while`-Schleife wie in Aufgabe 3.

Lösung

```
waermster = messwerte[0]
tag = 1
i = 0

while i < len(messwerte):
    if messwerte[i] > waermster:
        waermster = messwerte[i]
        tag = i + 1
    i = i + 1

print("Waermster:", waermster, "an Tag", tag)
```

Beide Zeilen stehen **im** `if`: Wert und Tag gehören zusammen und müssen gemeinsam wechseln. Steht `tag = i + 1` daneben statt darin, zeigt es am Ende auf den letzten Tag der Woche.

S021

S023

Aufgabe 6 – Wenn die Messung schiefgeht

Einstieg

Ein Sensor liefert nicht immer einen Messwert. Geht etwas schief, steht in der Reihe eine Fehlermarke – hier die -99. Öffne `messfehler.py`:

```
# Ein Sensor liefert nicht immer einen Messwert. Geht etwas schief,  
# schreibt er stattdessen eine Fehlermarke in die Reihe - hier die -99.  
  
messwerte = [12.4, -99, 18.9, 21.8, -99, 14.7, 23.5]  
  
for wert in messwerte:  
    print(wert)
```

Zähle mit einer Schleife, wie viele **gültige** Werte darin stehen, und gib die Zahl aus.

Lösung

```
anzahl = 0  
  
for wert in messwerte:  
    if wert != -99:  
        anzahl = anzahl + 1  
  
print("Gültige Werte:", anzahl)
```

`!=` heißt "ist ungleich". Fünf von sieben sind gültig.

Standard

Berechne den Mittelwert – aber nur aus den gültigen Werten. Die Fehlermarken dürfen das Ergebnis nicht verfälschen.

`sum` und `len` helfen hier nicht: Sie rechnen die -99 mit. Du brauchst zwei eigene Zähler – einen für die Summe, einen für die Anzahl.

Lösung

```
summe = 0  
anzahl = 0  
  
for wert in messwerte:  
    if wert != -99:  
        summe = summe + wert  
        anzahl = anzahl + 1  
  
print("Im Mittel:", round(summe / anzahl, 1))
```

Mit `sum(messwerte) / len(messwerte)` käme 2.5 heraus – die beiden -99 ziehen den Schnitt weit ins Minus.

Experte

Für Schleifen gibt es zwei Wörter, die den Ablauf abkürzen:

Zwei neue Wörter

`continue` bricht den **aktuellen Durchlauf** ab und springt zum nächsten Wert.

`break` verlässt die **ganze Schleife** – der Rest der Liste wird nicht mehr angesehen.

- a) Schreib deine Lösung von oben mit `continue`: Bei einer Fehlermarke sofort zum nächsten Wert springen, statt den Rest einzurücken.
- b) Schreib ein zweites Programm, das die Reihe durchgeht und beim **ersten** Wert über 23 Grad Alarm meldet und aufhört.

Lösung

```
summe = 0
anzahl = 0

for wert in messwerte:
    if wert == -99:
        continue
    summe = summe + wert
    anzahl = anzahl + 1
```

Der Rumpf rückt eine Stufe nach links: Was nach dem `continue` steht, gilt nur noch für gültige Werte.

```
for wert in messwerte:
    if wert > 23:
        print("Alarm bei", wert)
        break
```

Ohne `break` lief die Schleife weiter und meldete jeden weiteren Ausreißer mit. Gefragt war der erste.

S022

S024

Anhang • Listen, Verzweigungen und Schleifen

Alles von den Kärtchen noch einmal, zum Mitnehmen und Nachschlagen. Auswendig lernen musst du davon nichts.

Listen

Was	Beispiel	Ergebnis
anlegen	<code>werte = [12.4, 15.1, 18.9]</code>	
einzelnen Wert lesen	<code>werte[0]</code>	12.4
letzten Wert lesen	<code>werte[-1]</code>	18.9
Anzahl	<code>len(werte)</code>	3
anhängen	<code>werte.append(21.8)</code>	
zusammenzählen	<code>sum(werte)</code>	46.4
größter, kleinster	<code>max(werte), min(werte)</code>	

Gezählt wird ab null: `werte[0]` ist der erste Wert.

Verzweigen

```
if <Bedingung>:  
    <if-Rumpf>  
elif <Bedingung>:  
    <elif-Rumpf>  
else:  
    <else-Rumpf>
```

`elif` und `else` sind freiwillig. Doppelpunkt und Einrückung sind es nicht: Ohne Doppelpunkt kommt ein `SyntaxError`, ohne Einrückung ein `IndentationError`.

Wiederholen

```
while <Bedingung>:  
    <Rumpf>  
  
for <Name> in <Liste>:  
    <Rumpf>
```

`while` wiederholt, solange die Bedingung zutrifft – du musst selbst mitzählen: Startwert davor, Erhöhen im Rumpf. Fehlt das Erhöhen, läuft die Schleife endlos; stoppen mit dem roten **Stop**-Knopf.

`for` läuft einmal je Wert der Liste. Den Namen suchst du dir aus, er trägt bei jedem Durchlauf den nächsten Wert.

In beiden Schleifen gilt: `continue` bricht den aktuellen Durchlauf ab und springt zum nächsten, `break` verlässt die Schleife ganz.

Warten

```
import time  
  
time.sleep(0.5)
```

`import time` steht einmal ganz oben im Programm, `time.sleep` überall dort, wo eine Pause hingehört. Die Zahl sind Sekunden.