

Inhaltsverzeichnis

| | |
|---|----------|
| 1: Einleitung, erste Schritte, Sprachstruktur, interaktive Konsole | 2 |
| Einleitung | 2 |
| Erste Schritte | 2 |
| Die Sprachstruktur | 3 |
| REPL, die interaktive Konsole | 3 |
| 2. Kommentare, Blöcke und Operatoren | 4 |
| Kommentare | 4 |
| Blöcke | 4 |
| Operatoren | 5 |

1: Einleitung, erste Schritte, Sprachstruktur, interaktive Konsole

Einleitung

Dieser Kurs ist als Ergänzung zum Kurs 'Micropython mit ESP32' gedacht. Er gibt eine systematische Einführung in die Sprache Micropython.

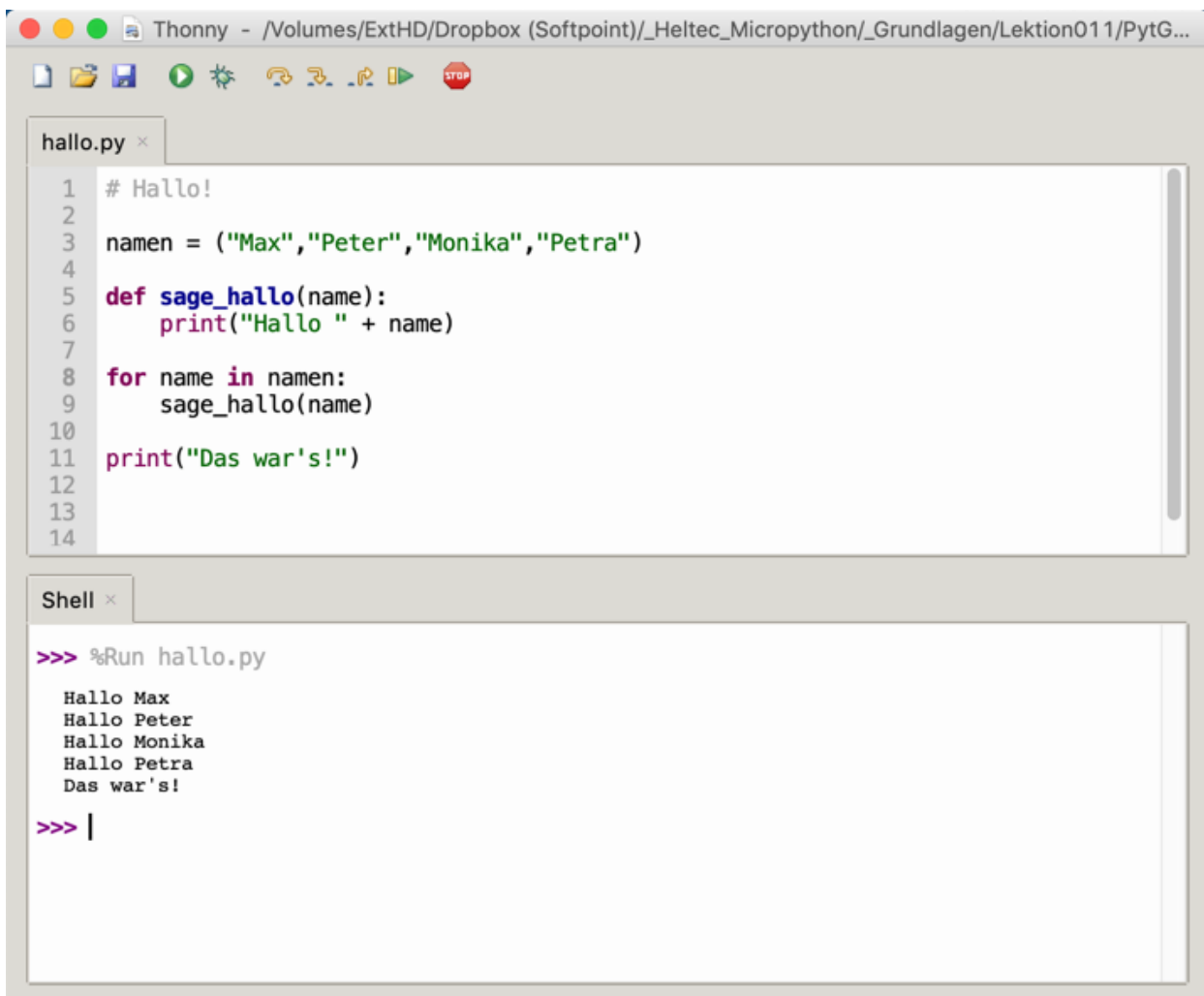
Ich nehme an, dass du die ersten Schritte mit dem ESP32 bereits unternommen hast. Damit sollten Thonny und das Heltec - Board bereits eingerichtet sein. Für diesen Kurs kannst du auch ein beliebiges anderes ESP32 - Board verwenden, falls du in der Lage bist, dieses einzurichten.

Vor Beginn jeder Lektion solltest du das Begleitmaterial herunterladen. Du findest den Link dazu in der Youtube Videobeschreibung oder im entsprechenden Eintrag im Forum.

Darin findest du alle Programmbeispiele (nur die Programme, nicht das was wir direkt in der Konsole eintippen) und die aktuellste Version dieses Dokumentes.

Erste Schritte

Das erste Programm ist sehr einfach. Erzeuge in Thonny ein neues Dokument und speichere es unter dem Namen `hallo.py` auf deiner Festplatte ab. Du kannst es aber auch direkt aus dem Begleitmaterial laden. Achte beim Abschreiben auf Gross- und Kleinschreibung. Das Einrücken nach dem `:` macht Thonny selbstständig. Du solltest das so lassen, da sonst das Programm nicht mehr läuft.



The screenshot shows the Thonny IDE interface. The top window, titled 'hallo.py', contains the following Python code:

```

1  # Hallo!
2
3  namen = ("Max", "Peter", "Monika", "Petra")
4
5  def sage_hallo(name):
6      print("Hallo " + name)
7
8  for name in namen:
9      sage_hallo(name)
10
11 print("Das war's!")
12
13
14


```

The bottom window, titled 'Shell', shows the execution of the script:

```

>>> %Run hallo.py
Hallo Max
Hallo Peter
Hallo Monika
Hallo Petra
Das war's!
>>> |

```

Es ist nicht notwendig, dass du das Programm vor der Ausführung auf dein Board speicherst. Ausführen kannst du das Programm durch Drücken auf den grünen Pfeil. 

Die Sprachstruktur

Dieses Beispiel gibt und die Gelegenheit einen ersten Blick auf die Sprachstruktur von Micropython zu werfen. Im Wesentlichen handelt es sich um Python 3. Da Mikrocontroller aber wesentlich weniger Speicher und Leistung besitzen als Desktopcomputer, wurden einige Anpassungen vorgenommen. Detaillierte Informationen findest du auf der [Webseite von Micropython](#).

Selbstverständlich unterstützt auch Micropython **Kommentare**. Diese werden hier mit # eingeleitet.

```
namen = ("Max", "Peter", "Monika", "Petra")
```

So werden Variablen angelegt. Sie entstehen durch Zuweisung eines Wertes. Dadurch erhalten sie auch gleich ihren Typ.

Als Nächsten finden wir eine Funktionsdefinition. Hier fällt der : und die Einrückung danach auf. Mit : starten wir einen Block (in C wäre das {). Der Block bleibt solange bestehen, wie wir die Einrückung beibehalten.

```
def sage_hallo(name):  
    print("Hallo " + name)
```

Ausserhalb des Blocks geht es mit

```
for name in namen:  
    sage_hallo(name)
```

weiter. Auch hier haben wir einen Block.

Das Ganze beenden wir mit dem Schlusssatz.

```
print("Das war's!")
```


In Python wird normalerweise nur ein Befehl pro Zeile geschrieben. Der Befehl muss auch nicht mit ; abgeschlossen werden.

REPL, die interaktive Konsole

REPL steht für read-evaluate-print loop. Diese Konsole kann Befehle entgegennehmen, ausführen und Ergebnisse ausgeben.

Python ist eine interpretierte Sprache. Wir können jeden Befehl direkt eingeben und ausführen lassen. Es ist nicht notwendig, die Befehle zuerst in Maschinensprache zu übersetzen.

Hier einige Beispiele:



```
Shell <
>>> 4 + 5
9
>>> print(4 * 5)
20
>>> s = "25 / 3"
>>> print(s)
25 / 3
>>> print(eval(s))
8.333333
>>> |
```

2. Kommentare, Blöcke und Operatoren

Kommentare

In jedem Programm sind erläuternde Kommentare wichtig. Nur so weiss man nach Jahren noch, was man sich damals dabei gedacht hat.

Es gibt einfache Kommentare, die immer mit # eingeleitet werden. Sie beginnen mit # und enden mit dem Zeilenende.

```
a = 5 # Die Variable a wird auf 5 gesetzt
```

Mehrzeilige Kommentare werden mit """ oder ''' eingeleitet. Sie werden mit derselben Zeichenfolge beendet.

```
"""
erste Kommentarzeile
zweite Kommentarzeile
"""
```

oder

```
'''
erste Kommentarzeile
zweite Kommentarzeile
'''
```

Blöcke

In der Sprache C werden Blöcke bekanntlich in {} eingeschlossen. Python verwendet hier ein etwas anderes Konzept. Einrückungen, die in C nur der Übersichtlichkeit dienen, sind in Python Bestandteil der Syntax und helfen, Blöcke zu definieren.

```
def addiere(a,b):    # Mit dem : wird der Block eingeleitet
    c = a + b        # Nach dem : wird eingerückt
    print(c)         # Diese Einrückung bleibt für alle Befehle
                    # innerhalb des Blockes erhalten

addiere(3,5)         # Keine Einrückung mehr! Damit gehört dieser Befehl
                    # nicht mehr zum Block
```

In Bedingungen und Schleifen werden Blöcke auf identische Art gebildet.

Operatoren

Arithmetische Operatoren

| | | |
|------|----------------------------|----------------------------|
| +, - | Addition Subtraktion | $c = a + b$ $c = a - b$ |
| *, / | Multiplikation Division | $c = a * b$ $c = a / b$ |
| // | Ganzzahlige Division | $11 // 3 ==> 3$ |
| % | Modulo (Rest der Division) | $11 \% 3 ==> 2$ |
| ** | Exponent | $2 ** 3 ==> 8$ |

Logische Operatoren

and, or, not Boolesche Operatoren

Bitoperatoren

| | |
|----|------------------------------|
| & | Bitweise AND - Verknüpfung |
| | Bitweise OR - Verknüpfung |
| ^ | Bitweise XOR - Verknüpfung |
| ~ | Bitweises NOT |
| << | Bits nach links verschieben |
| >> | Bits nach rechts verschieben |

Vergleiche

| | | |
|--------|------------------------|--------------------------------|
| == | ist gleich | $3 == 5$ # das ergibt False |
| != | ist ungleich | $3 != 5$ # das ergibt True |
| > | grösser | $3 > 5$ # das ergibt False |
| < | kleiner | $3 < 5$ # das ergibt True |
| <= | kleiner oder gleich | |
| >= | grösser oder gleich | |
| in | ist enthalten in | "Max" in ("Max", "Peter") |
| not in | ist nicht enthalten in | "Fritz" in ("Monika", "Petra") |