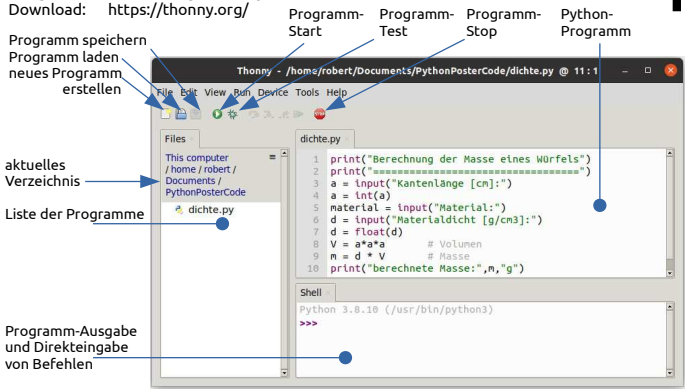
Python - die Grundlagen im Überblick

Programmierwerkzeug: Thonny … ist beim Raspberry Pi bereits installiert
Download: https://thonny.org/



1 Anzeigen, Zahleneingaben, Berechnungen

print zeigt Texte an:
`print("Hello World")`

Variablen speichern Werte. Dazu ist eine Wertezuweisung nötig:
`a = 10`
Es kann aber auch eine Formel programmiert werden und das Berechnungsergebnis wird in der Variablen gespeichert:
`m = d * v`
Die Werte der Variablen können mit print angezeigt werden:
`print("Masse:",m)`

Berechnung der Masse eines Goldwürfels

```
a = 10          # Seitenlänge in cm
d = 19.3        # Dichte Gold: 19,3 g/cm3
v = a*a*a       # Volumen
m = d * v       # Masse
print("Goldwürfel mit Seitenlänge",a,"cm")
print("Masse:",m,"g")
```

input übergibt Eingaben an Variable:
`text = input("Wie geht es dir?")`

Jede Tastatureingabe von input ist stets ein Text. Soll eine Zahl eingegeben werden, um damit zu rechnen, so muss extra eine Umwandlung in den gewünschten Zahlentyp erfolgen.

```
a = input("Kantenlänge [cm]:")
a = int(a)

d = input("Dichte [g/cm3]:")
d = float(d)
```

int … Umwandlung in ganze Zahl
float … Umwandlung in reelle Zahl

```
print("Berechnung der Masse eines Würfels")
print("================================")
a = input("Kantenlänge [cm]:")
a = int(a)
material = input("Material:")
d = input("Materialdichte [g/cm3]:")
d = float(d)
v = a*a*a       # Volumen
m = d * v       # Masse
print("berechnete Masse:",m,"g",material)
```

2 Bedingungen

if vergleicht Werte und reagiert entsprechend. Wenn die Bedingung erfüllt ist, werden die nachfolgenden Befehle ausgeführt. Diese Befehle sind als Anweisungsblock vier Leerzeichen eingerückt!

```
if a>0:
    v = a*a*a
    print(v)
```

else nach if enthält Befehle, die ausgeführt werden, wenn die if-Bedingung nicht gilt:

```
if a>0:
    v = a*a*a
    print(v)
else:
    print("Ungültige Kantenlänge.")
    print("Programmabbruch!")
    exit(0)
```

exit(0) … beendet das Programm.

```
print("Berechnung der Masse eines Würfels")
print("================================")
a = input("Kantenlänge [cm]:")
a = int(a)
mat = input("Material:")
d = input("Materialdichte [g/cm3]:")
d = float(d)
if a>0 and d>0:
    v = a*a*a       # Volumen
    m = d * v       # Masse
    print("berechnete Masse:",m,"g",mat)
else:
    print("Ungültige Werte.")
```

Hinweise zu if-Anweisungen

Die if- und die else-Zeile endet immer mit **:**

Die else-Programmierung muss nicht erfolgen.

Mehrfachvergleiche mit and / or:
… and … beide Vergleiche müssen gelten
… or … es reicht wenn, ein Vergleich gilt

mögliche Vergleiche:
`<` kleiner als
`>` größer als
`<=` kleiner oder gleich als
`>=` größer oder gleich als
`==` gleich
`!=` ungleich

3 Wiederholungen / Schleifen

while wiederholt Befehle, solange eine Ver-gleichsbedingung gilt. Alle Befehle, die durch ein while zu wiederholen sind, werden um vier Leerzeichen eingerückt.

Im folgenden Beispiel wird die Berechnung und Anzeige solange wiederholt, wie der Wert von a in jeder Schleifenausführung um 10 erhöht.

```
a = 10
while a<=100:
    v = a*a*a
    print(v)
    a=a+10
print("Ende der Schleife")
```

```
print("Berechnung der Masse eines Würfels")
print("================================")
print("Kantenlänge in cm")
a = input("Startwert [cm]:")
a = int(a)
b = input("Endwert [cm]:")
b = int(b)
mat = input("Material:")
d = input("Materialdichte [g/cm3]:")
d = float(d)
while a<=b:
    v = a*a*a       # Volumen
    m = d * v       # Masse
    print(a,"cm ... Masse:",m,"g",mat)
    a=a+1
```

while True: programmiert eine Schleife, die solange läuft, bis sie mit **break** abgebrochen wird. Dafür wird die break-Anweisung mit einer if-Anweisung ausgelöst. Bei einem Abbruch mit break wird das Programm an der ersten Zeile nach der Schleife fortgesetzt.

```
print("Berechnung der Masse eines Würfels")
print("================================")
mat = input("Material:")
d = input("Materialdichte [g/cm3]:")
d = float(d)
while True:
    a = input("Kantenlänge [cm]:")
    a = int(a)
    if a<0:         # unsinnige Länge?
        break       # Abbruch
    v = a*a*a       # Volumen berechnen
    m = d * v       # Masse berechnen
    print(a,"cm ... Masse:",m,"g",mat)
print("Programmende.")
```

4 Die Liste als Datenspeicher

Liste
Eine Liste speichert eine Folge von Werten. Die Werte werden in eckigen Klammern, durch Komma getrennt angegeben.

```
print("Berechnung der Masse eines Würfels")
print("================================")

material = ["Gold","Silber","Messing"]
dichte = [19.3, 10.5, 8.4]
```

Die **for-in-Schleife** liefert nacheinander jeden Listeneintrag.

```
print("-Materialübersicht-")
for i in material:
    print(i)

a = input("Kantenlänge [cm]:")
a = int(a)
v = a*a*a
```

Die Funktion **len** liefert die Anzahl der Listeneinträge.

Der Index ist die Positionsnummer eines Elements in der Liste. Das erste Element hat den Index 0.

Mit dem **Index in eckigen Klammern** wird auf das Element an der jeweiligen Listenposition zugegriffen.

```
anzahl = len(material)
index = 0
while index<anzahl:
    mat = material[index]
    d = dichte[index]
    masse = d * v
    print(mat," ... Masse:",masse,"g")
    index = index+1
print("Programmende.")
```

5 Das Dictionary als Datenspeicher

Ein **Dictionary** speichert Paare **Schlüsselwort : Wert**.
Die Paare werden in geschweiften Klammern durch Komma getrennt angegeben.
Die Operation **keys()** liefert die Menge aller Schlüsselworte.

```
print("Berechnung der Masse eines Würfels")
print("================================")

dichte = {"Gold":19.3,"Silber":10.5,"Blei":11.3}
material = dichte.keys()

print("-Materialübersicht-")
for i in material:
    print(i)
```

Mit der Operation *Element* **in** *Menge* wird geprüft, ob ein Element in einer Menge oder Liste enthalten ist.

Die Operation **get(Schlüsselwort)** liefert den Wert, der zu diesem Schlüsselwort im Dictionary gespeichert ist.

```
mat = input("Material:")
if mat in material:
    a = input("Kantenlänge [cm]:")
    a = int(a)
    v = a*a*a
    d = dichte.get(mat)
    masse = d * v
    print(mat," ... Masse:",masse,"g")
else:
    print("Material nicht vorhanden.")
```

6 Funktionen erledigen Aufgaben

def … Beginn einer Funktionsdefinition
Funktionsname Parameter

Funktionen sind Programmbausteine, die eine bestimmte Aufgabe erledigen. Meist sind das Berechnungen oder Anzeigen von Daten.
Eine Funktion hat einen Namen und einen oder mehrere Parameter. Der Code einer Funktion wird um vier Leerzeichen eingerückt. Er realisiert die Berechnungen oder die Datenanzeige.
Die **return**-Anweisung gibt das Ergebnis der Funktion zurück.

Variablen, die innerhalb von Funktionen definiert werden, gelten nur innerhalb der Funktion. Veränderungen, die innerhalb einer Funktion an einem Dictionary oder an einer Liste vorgenommen werden, sind nach Ende der Funktion überall im Programm wirksam.

Funktionen können beliebig oft an verschiedenen Stellen im Programm aufgerufen werden.

```
def berechneMasse(seite, dichte):
    v = seite**3
    masse = dichte * v
    return masse

def zeigeMaterial(material):
    print("-Materialübersicht-")
    for i in material:
        print(i)

print("Berechnung der Masse eines Würfels")
print("================================")

dichte = {"Gold":19.3,"Silber":10.5,"Blei":11.3}
material = dichte.keys()

zeigeMaterial(material)
mat = input("Material:")
if mat in material:
    a = input("Kantenlänge [cm]:")
    a = int(a)
    d = dichte.get(mat)
    masse = berechneMasse(a,d)
    print(mat," ... Masse:",masse,"g")
```

7 Bibliotheken und Diagramme

Bibliotheken oder Module stellen viele nützliche Funktionen zur Verfügung. Die benötigten Module werden mit **import** am Beginn des Programms eingebunden. Danach können die Funktionen dieser Module im Programm benutzt werden.

Das Modul math enthält u. a. die Winkelfunktionen (sin, cos, …) und die Wurzelfunktion.

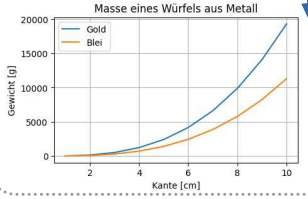
Das Modul **pyplot** bietet Funktionen zum Erzeugen von Diagrammen.

```
import matplotlib.pyplot as plt

def berechneMasse(seite, dichte):
    v = seite**3
    masse = dichte * v
    return masse

print("Berechnung der Masse eines Würfels")
print("================================")

dichte = {"Gold":19.3,"Silber":10.5,"Blei":11.3}
print(dichte.keys())
mat1 = input("Material 1:")
mat2 = input("Material 2:")

if mat1 not in dichte.keys():
    exit(0)
if mat2 not in dichte.keys():
    exit(0)

d1= dichte.get(mat1)
d2= dichte.get(mat2)
a=1
gewicht1 = []
gewicht2 = []
kante = []
while a<=10:
    masse1 = berechneMasse(a,d1)
    masse2 = berechneMasse(a,d2)
    gewicht1.append(masse1)
    gewicht2.append(masse2)
    kante.append(a)
    a=a+1
```

3 leere Listen erzeugen

Daten in Listen einfügen

```
Berechnung der Masse eines Würfels
==================================
dict_keys(['Gold', 'Silber', 'Blei'])

Material 1: Gold
Material 2: Blei
```




```
fig, ax = plt.subplots(figsize=(5, 3))
ax.plot(kante, gewicht1, label=mat1)
ax.plot(kante, gewicht2, label=mat2)
ax.set_title("Masse eines Würfels aus Metall")
ax.set_xlabel('Kante [cm]')
ax.set_ylabel('Gewicht [g]')
ax.grid(True) # zeige Gitter
ax.legend()   # zeige Legende.
plt.show()
```

Zahlenwerte der Listen als Diagramm anzeigen

• als Dezimaltrennzeichen und nicht Komma
markiert Beginn von Kommentartext

Beachte bei if, else, while, for, def:
: Doppelpunkt am Ende nicht vergessen
4 Leerzeichen rücken Anweisungsblöcke ein

HTWD Hochschule für Technik und Wirtschaft Dresden University of Applied Sciences
Informatik studieren
Kontakt: robert.ringel@htw-dresden.de