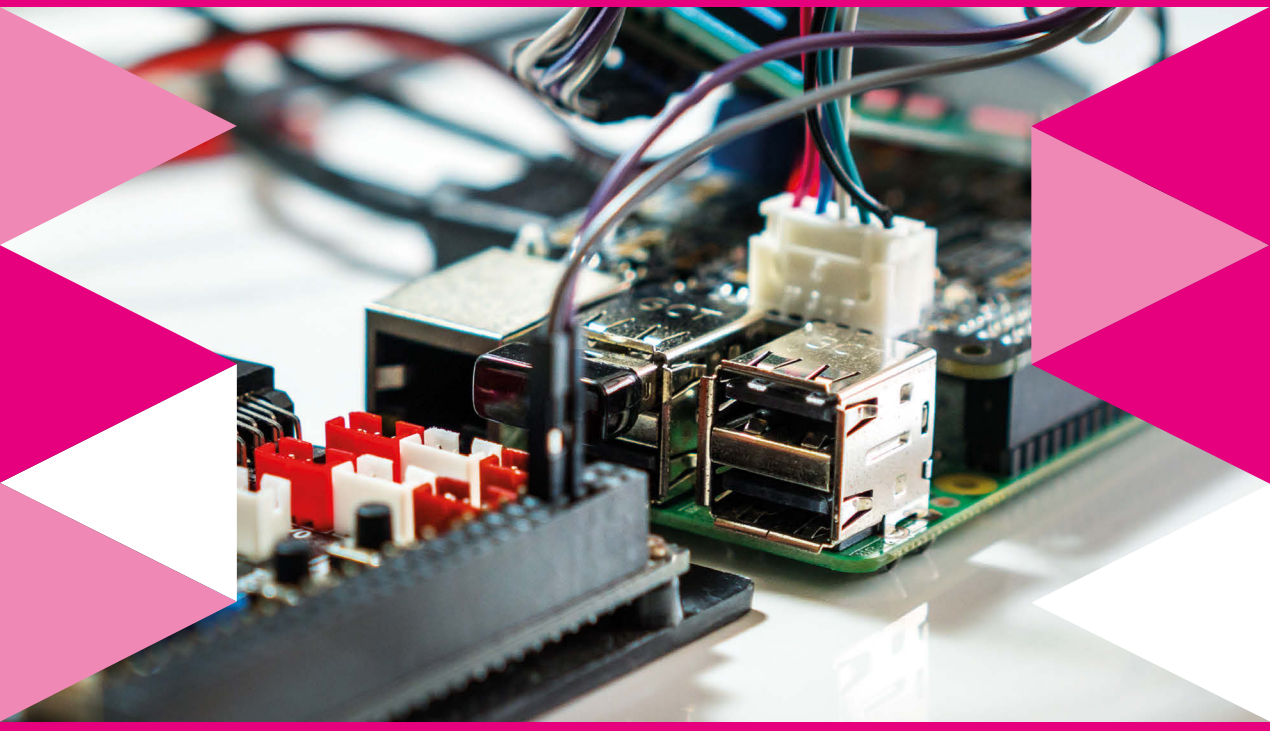
FRANZIS

MACH'S EINFACH

Erste Schritte RASPBERRY PI PROGRAMMIEREN

Der perfekte Einstieg in die Programmierung mit Scratch und Python



CHRISTIAN IMMLER

Erste Schritte

RASPBERRY PI PROGRAMMIEREN

Der perfekte Einstieg in die Programmierung mit Scratch und Python

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte Daten sind im Internet über *http://dnb.ddb.de* abrufbar.

Der Autor
Christian Immler, Jahrgang 1964, war bis 1998 als Dozent für Computer Aided Design an der Fachhochschule Nienburg und an der University of Brighton tätig. Einen Namen hat er sich mit diversen Veröffentlichungen zu Spezialthemen wie 3D-Visualisierung, PDA-Betriebssysteme, Linux und Windows gemacht. Seit mehr als 15 Jahren arbeitet er als erfolgreicher Autor mit mehr als 20 veröffentlichten Computerbüchern.

Lektorat: Ulrich Dorn
Satz: Nelli Ferderer (nelli@ferderer.de)
Covergestaltung: Julia Harrer
ISBN 978-3-645-20626-6

Kaum ein elektronisches Gerät in seiner Preisklasse hat in den letzten Jahren so viel von sich reden gemacht wie der Raspberry Pi. Ursprünglich als Computer für den Schulunterricht geplant, hat die Maker-Szene ganz schnell Spaß daran gefunden.

Der Raspberry Pi ist – auch wenn es auf den ersten Blick gar nicht so aussieht – ein vollwertiger Computer, etwa in der Größe einer Kreditkarte und vor allem zu einem sehr günstigen Preis. Nicht nur die Hardware ist günstig, die Software noch mehr. Das Betriebssystem und alle im Alltag erforderlichen Anwendungen werden kostenlos zum Download angeboten.

Der Raspberry Pi eignet sich besonders gut zum Einstieg in die Programmierung, da die kompletten Entwicklungsumgebungen für zwei leicht zu erlernende Programmiersprachen betriebsbereit installiert sind. Mit der auf dem Raspberry Pi eingebauten GPIO-Schnittstelle kann man auch direkt angeschlossene Elektronik über eigene Programme ansteuern, was auf einem PC nur mit erheblichem Aufwand möglich ist.

- **Scratch** ist eine intuitive Programmierumgebung, mit der Kinder und Programmiereinsteiger schnell Ideen umsetzen können, ohne sich in zuerst mit Programmiertheorie auseinandersetzen zu müssen. Die Programe werden intuitiv aus Bausteinen zusammengesetzt, man muss sich nicht erst mit der Syntax der Sprache vertraut machen.
- **Python** ist zum Einstieg in die Programmierung auf dem Raspberry Pi vorinstalliert und gab ihm auch einen Teil seines Namens. Python überzeugt durch seine klare Struktur, die einen einfachen Einstieg in das Programmieren erlaubt, ist aber auch eine ideale Sprache, um »mal schnell« etwas zu automatisieren, was man sonst von Hand erledigen würde.

Die Symbole im Buch

Auffällige Symbole helfen beim Überblick über die vielfältigen Informationen in diesem Buch – besonders wenn Sie später noch einmal etwas nachschlagen möchten.

- **Wichtige Schritte**, die erledigt werden müssen, damit etwas funktioniert oder einfach wichtiges Wissen, das man zum Verständnis braucht.

- **Elektroniktipps** zu Bauteilen und Schaltungen.

- **Downloads** – einfach herunterladen, statt abzutippen oder zu googeln.

- Nützliche Tipps zum Raspberry-Pi-**Desktop** – nicht alles ist so, wie man es von Windows kennt.

- **Kommandozeile** – man braucht die Kommandozeile in Linux nur noch selten, aber hilfreich ist sie dennoch.






- **Einstellungen** – im Betriebssystem, auf der grafischen Oberfläche oder in Programmen.
- **Python** – wichtige Programmiertricks, Syntaxregeln oder Definitionen, die man zum Programmieren in Python immer wieder braucht.
- **Insiderwissen** oder was man sich früher in einem Buch rot angestrichen hätte.
- **Warten** – Hier kann es mal etwas länger dauern.
- **Vorsicht** – Hier kann etwas schiefgehen!
- **Download** – Alle Code-Beispiele und weitere nützliche Dateien finden Sie gratis zum Download unter *www.buch.cd*.

Raspberry Pi vorbereiten

Um den Raspberry Pi in Betrieb zu nehmen, braucht man:

- USB-Tastatur und Maus
- HDMI-Kabel für Monitor
- Netzwerkkabel oder WLAN
- MicroSD-Karte mit Betriebssystem Raspbian
- Micro-USB-Handyladegerät als Netzteil (mindestens 1.500 mA)
- Audiokabel für Lautsprecher (optional)

Das Netzteil muss als Letztes angeschlossen werden, damit schaltet sich der Raspberry Pi automatisch ein. Es gibt keinen eigenen Ein-/Ausschalter.

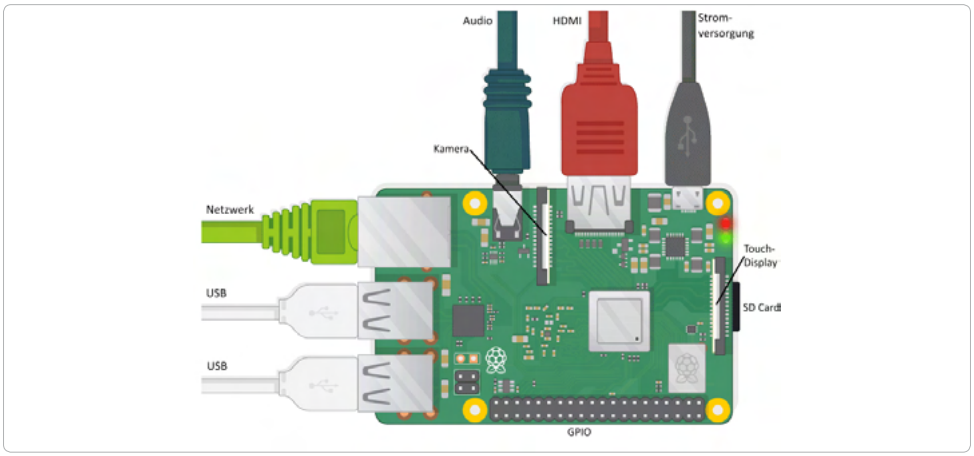


Bild 1: Die Anschlüsse am Raspberry Pi 3 B+ (Grafik: Raspberry Pi Foundation – Creative Commons Lizenz).

Der neue Raspberry Pi 4

Im Juni 2019 präsentierte die Raspberry Pi Foundation den neuen Raspberry Pi 4 mit deutlich leistungsfähigerer Hardware.

- Erstmals gibt es den Raspberry Pi mit drei verschiedenen Arbeitsspeichergrößen: 1 GB, 2 GB und 4 GB.
- Prozessor: Broadcom BCM2711, Quad core Cortex-A72 64-bit SoC 1.5 GHz.
- Netzwerk: Gigabit Ethernet und Dual-Band IEEE 802.11ac WLAN
- 2x USB 3.0, 2x USB 2.0 (an der Farbe unterscheidbar)
- 2x Micro-HDMI-Anschlüsse. Erstmals werden zwei Monitore unterstützt, zum Anschluss sind Micro-HDMI-Kabel oder Adapter erforderlich.
- USB-Typ-C-Anschluss zur Stromversorgung, mindestens 3 A erforderlich

Der Steckplatz für MicroSD-Karten, die 40-polige GPIO-Pinleiste sowie die Klinkenbuchse für Stereo-Audio und Composite Video sind gegenüber dem Vorgängermodell unverändert.

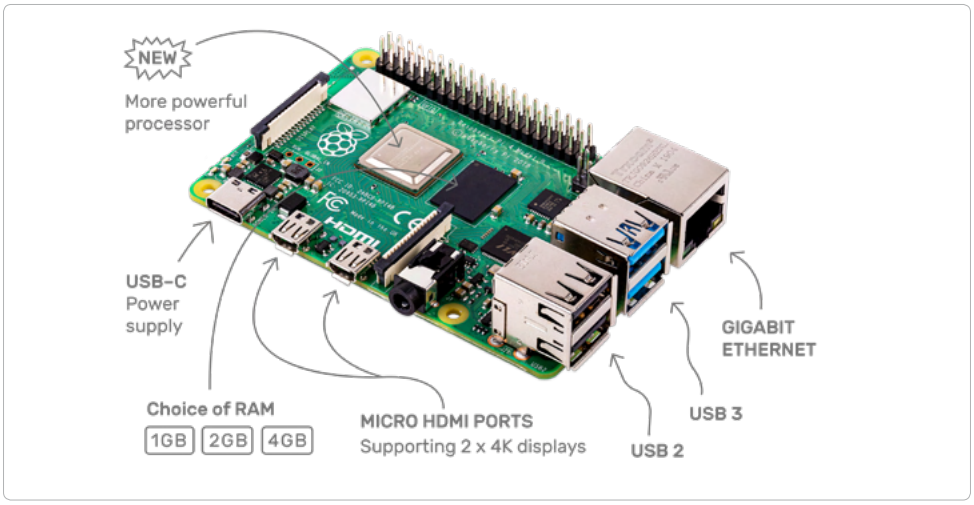


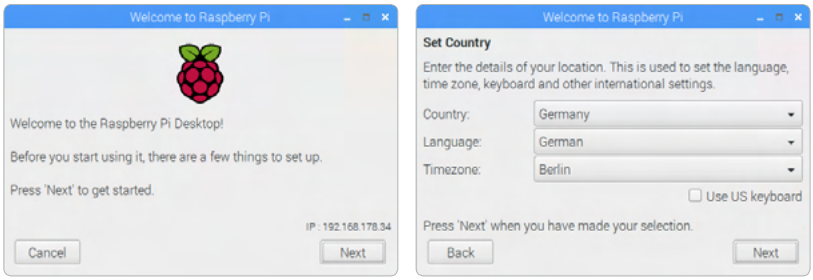
Bild 2: Die Anschlüsse des Raspberry Pi 4 (Grafik: Raspberry Pi Foundation – Creative Commons Lizenz).

Verwenden Sie für Tastatur und Maus die USB-2.0-Anschlüsse. Die USB-3.0-Anschlüsse verwenden Sie besser für USB-Sticks und externe Festplatten.

Betriebssysteminstallation in Kürze

Für alle, die ihren Raspberry Pi noch nicht mit der aktuellen Raspbian-Version betriebsbereit haben, folgt hier die Systeminstallation in zehn Schritten:

1. NOOBS (mindestens Version 3.1.1) von *www.raspberrypi.org/downloads* auf den PC herunterladen und Zip-Archiv auf die Festplatte entpacken.
2. Wurde die SD-Karte bereits benutzt, mit SD-Formatter im PC neu formatieren: *www.sdcard.org/downloads/formatter_4*. Dabei *Format Size Adjustment* einschalten (die SD-Karte muss mindestens 8 GB groß sein).
3. Alle Dateien und Unterverzeichnisse von NOOBS auf die SD-Karte kopieren.
4. SD-Karte aus dem PC nehmen, in den Raspberry Pi stecken und booten. Ganz unten *Deutsch* als Installationssprache wählen. Damit wird automatisch auch die deutsche Tastatur ausgewählt.
5. Das Häkchen beim vorausgewählten Raspbian-Betriebssystem setzen und oben links auf *Install* klicken. Nach Bestätigung einer Sicherheitsabfrage, dass die Speicherkarte komplett überschrieben wird, startet die Installation, die einige Minuten dauert.
6. Nach abgeschlossener Installation bootet der Raspberry Pi neu.
7. Auf dem Raspbian-Desktop startet der Konfigurationsassistent und zeigt die IP-Adresse des Raspberry Pi. Hier auf *Next* klicken und Sprache und Zeitzone auswählen, falls sie nicht automatisch auf Deutsch gesetzt sind.



8. Im nächsten Schritt wird empfohlen, das Standardpasswort zu ändern, was aber nicht zwingend nötig ist. Der Standardbenutzer *pi* wird beim Booten automatisch angemeldet, sodass Sie das Passwort nur selten brauchen werden.
9. Bei WLAN-Verbindung das gewünschte Netzwerk auswählen und das Passwort eingeben, bei Ethernetverbindung einfach auf *Skip* klicken.

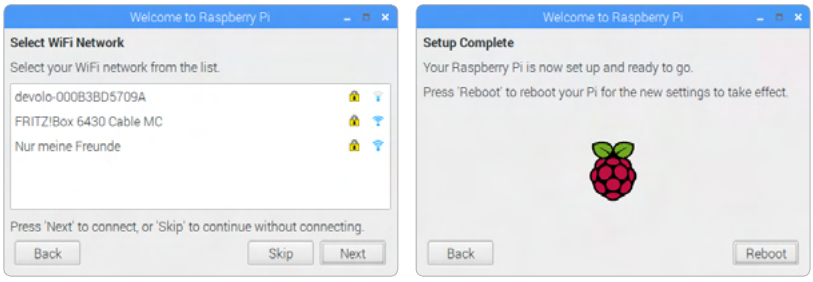


Bild 3: Zum Schluss automatisch Updates herunterladen und den Raspberry Pi neu starten.

Download der Beispielprogramme zum Buch

Zu diesem Buch bieten wir Zusatzmaterial zum Download, wie unter anderem alle Beispielprogramme aus den folgenden Kapiteln.

1. Besuchen Sie mit dem Chromium-Browser auf dem Raspberry Pi die Seite: *www.buch.cd*
2. Geben Sie dort diesen Code ein: 60626-4
3. Folgen Sie den Anweisungen zum Download. Der Chromium-Browser speichert die Zip-Datei standardmäßig unter `/home/pi/Downloads`. Übernehmen Sie diese Vorgabe.

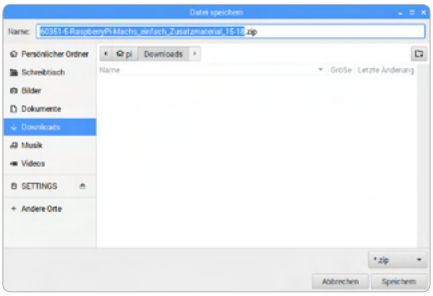


Bild 4: Downloadordner in Chromium wählen

4. Nach erfolgreichem Download klicken Sie unten links im Browser auf die heruntergeladene Datei und wählen *Öffnen*.
5. Die Datei wird im vorinstallierten **Xarchiver** geöffnet. Klicken Sie mit der rechten Maustaste auf den angezeigten Ordner und wählen Sie im Menü *Entpacken*. Tragen Sie im Feld *Entpacken nach* das Home-Verzeichnis `/home/pi` ein.
6. Klicken Sie auf *Entpacken*. Danach liegen die Dateien in Ihrem Home-Verzeichnis.

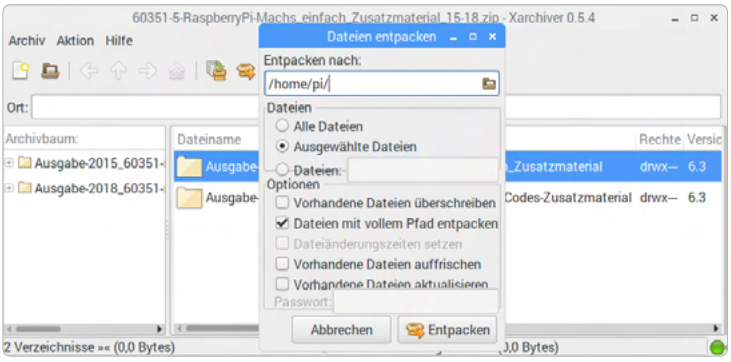


Bild 5: Heruntergeladenes Zip-Archiv entpacken

INHALT

Zum Einstieg in die Programmierung ist auf dem Raspberry Pi die Programmiersprache Python vorinstalliert. Python ist übersichtlich und relativ einfach zu erlernen. Da keine Variablendeklarationen, Typen, Klassen oder komplizierten Regeln zu beachten sind, macht das Programmieren wirklich Spaß.

Der neue Code-Editor Mu

Seit der Betriebssystemversion NOOBS 3.1.1, die speziell für den Raspberry Pi 4 entwickelt wurde, ist statt der klassischen Python-Entwicklungsumgebung IDLE der neue Code-Editor *Mu* im Menü *Entwicklung* vorinstalliert.

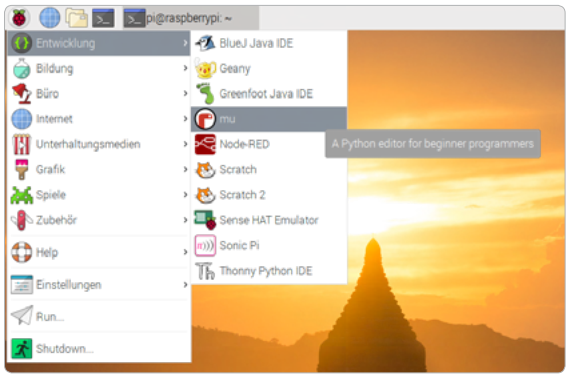


Bild 2.1: Der Code-Editor Mu im Menü.

Wählen Sie beim ersten Start von Mu den Modus Python 3. Über das Symbol *Modus* können Sie jederzeit auch wieder zu einer anderen Programmiersprache wechseln.

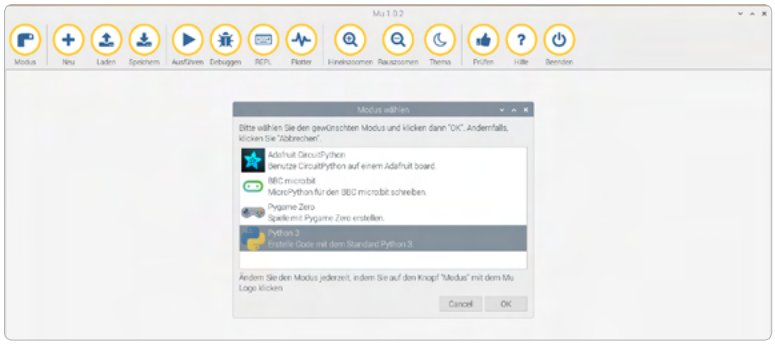


Bild 2.2: Der erste Start von Mu.

Mu zeigt sich im Python-Modus auf den ersten Blick als simples Eingabefenster zur Eingabe von Code.

Hallo Welt

Üblicherweise fangen Programmierkurse mit einem **Hallo-Welt**-Programm an, das auf den Bildschirm die Worte »Hallo Welt« schreibt. Ein »Hallo Welt« ist in Python derart einfach, dass es sich nicht einmal lohnt, dafür eine eigene Überschrift zu vergeben. Tippen Sie einfach folgende Zeile im Mu-Fenster ein:

```
print(Hallo Welt)
```

Dieses erste »Programm« schreibt *Hallo Welt* in das untere Teilfenster von Mu.

Bereits beim Tippen sehen Sie interaktive Hilfetexte. Mu erkennt die Wörter und Standardfunktionen der Programmiersprache Python und zeigt Tipps zur Verwendung an.

Bild 2.3: Tooltipp beim Schreiben von Python-Programmen in Mu.

Klicken Sie auf das Symbol *Ausführen*. Als Erstes müssen Sie einen Dateinamen angeben, unter dem das Programm gespeichert wird. Mu speichert Programme standardmäßig unter `/home/pi/mucode`. Sie können aber auch das Home-Verzeichnis `/home/pi` oder einen Unterordner davon verwenden.

Dateiendungen spielen in Linux zwar nur eine untergeordnete Rolle. Sie sollten sich aber trotzdem angewöhnen, Python-Programmen die Endung `.py` zu geben.

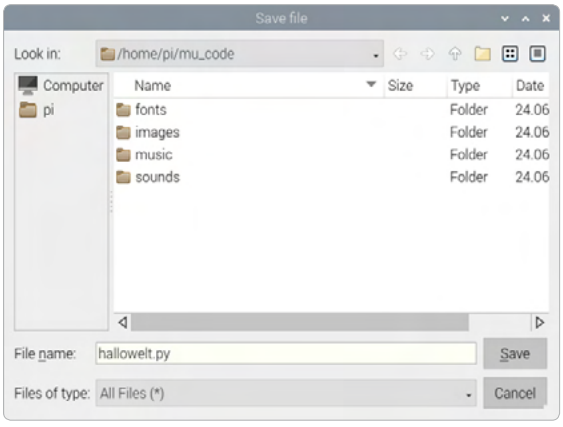


Bild 2.4: Programm als Datei mit der Endung .py speichern

Danach wird das Programm automatisch gestartet. Dazu erscheint unten im Mu-Fenster ein neues Teilfenster mit den Programmausgaben.

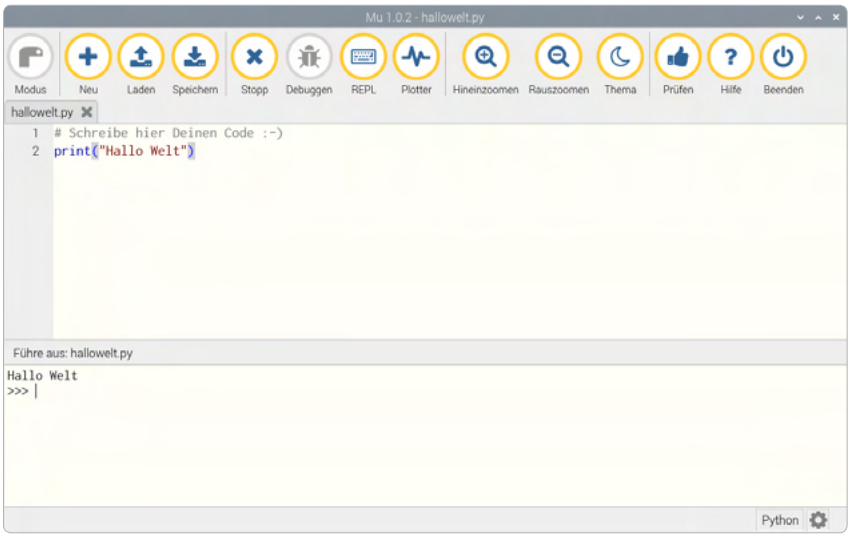


Bild 2.5: Hallo Welt in Python.

Ein Klick auf das Symbol *Stopp* schließt das untere Teilfenster wieder.

Die Zeile

```
# Schreibe hier Deinen Code :-)
```

ist wie alle Zeilen, die mit einem #-Zeichen beginnen, nur Kommentar und kann problemlos gelöscht werden.

Python 2 oder 3 interaktiv nutzen

Auf dem Raspberry Pi ist neben der neuen Python-Version 3 auch noch das klassische Python 2 vorinstalliert, das aber nicht innerhalb von Mu verwendet werden kann. Python 3 verwendet teilweise eine andere Syntax als die altbewährte Version 2.7.x, sodass Programme aus der einen Version nicht mit der anderen laufen. Lange Zeit waren wichtige Bibliotheken noch nicht für Python 3.x verfügbar. Inzwischen hat sich Python 3.x zum Standard entwickelt und wird auch für die Beispiele in diesem Buch verwendet. Python 2 wird weiterhin geringfügig weiterentwickelt, um Fehler oder Sicherheitslücken zu korrigieren.

Falls Sie die pure Python-Shell interaktiv und einfach, ohne die Kommandozeile aufzurufen, nutzen möchten, starten Sie im Startmenü unter *Einstellungen* den *Main Menu Editor* und schalten dort im Bereich *Entwicklung* die Häkchen bei *Python (v2,7)* und *Python (v3,7)* ein.

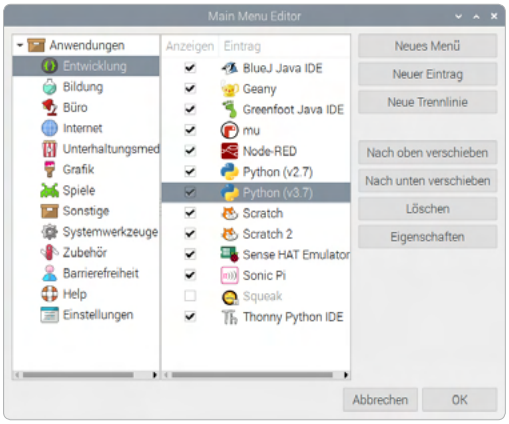


Bild 2.6: Python Shells im Startmenü aktivieren.

Jetzt können Sie die Python-Shells direkt über das Menü *Entwicklung* aufrufen.

In diesem Fenster können Sie Python-Kommandos direkt interaktiv abarbeiten, ohne ein eigentliches Programm schreiben zu müssen. Geben Sie zum Beispiel am Prompt von Python 3.7 Folgendes ein:

```
>>> 1+2*3
```

Sofort erscheint die richtige Antwort:

```
7
```

Bild 2.7: Das Eingabefenster der Python-Shell.

Python rechnet automatisch »Punkt vor Strich«. Auf diese Weise lässt sich Python als komfortabler Taschenrechner verwenden, was aber noch nichts mit Programmierung zu tun hat.

Bereits in diesem Taschenrechner-Modus zeigt sich ein wichtiger Unterschied zwischen Python 2 und Python 3, nämlich bei der Division zweier Integer-Zahlen.

Bild 2.8: Ganzzahldivision in Python 3.

Python 3 dividiert zwei Ganzzahlen immer mathematisch korrekt. Dabei kann als Ergebnis eine Fließkommazahl herauskommen. Der Operator `//` bewirkt eine echte Ganzzahldivision mit ganzzahligem Ergebnis.

Python 2 dividiert dagegen zwei Ganzzahlwerte automatisch mit Ganzzahldivision. Dabei ergibt sich immer auch ein ganzzahliges Ergebnis. Um ein mathematisch korrektes Ergebnis zu erzielen, muss mindestens einer der Werte als Fließkommazahl eingegeben werden.

Bild 2.9: Ganzzahldivision in Python 2.

DEZIMALTRENNZEICHEN

Python verwendet wie viele britische oder amerikanische Programme den Punkt als Dezimaltrennzeichen, nicht das in Deutschland übliche Komma. Einhalb wird also als `0.5` angegeben und nicht als `0,5`.

Wichtige Operatoren in Python

Die Operatoren zum Rechnen mit Zahlen sind in Python ähnlich wie in anderen Programmiersprachen.

OPERATOR	BEDEUTUNG	BEISPIEL	ERGEBNIS
+	Addition	1 + 2	3
-	Subtraktion	3 – 1	2
*	Multiplikation	2 * 3	6
**	Exponentiation	3 ** 2	9
/	Division	3 / 2	1.5
//	Ganzzahldivision	3 // 2	1
%	Modulo (ganzzahlig unteilbarer Rest)	5 % 3	2
<	Kleiner	2 < 3	True
<=	Kleiner oder gleich	2 <= 2	True
>	Größer	2 > 3	False
>=	Größer oder gleich	3 >= 3	True
==	Gleich	2 == 1 + 1	True
!=	Ungleich	2 != 1 + 1	False

Die Python-Flashcards

Python ist die ideale Programmiersprache, um den Einstieg in die Programmierung zu erlernen. Nur die Syntax und die Layoutregeln sind etwas gewöhnungsbedürftig. Zur Hilfestellung im Programmieralltag werden die wichtigsten Syntaxelemente der Sprache Python auf den folgenden Seiten in Form kleiner »Spickzettel« kurz beschrieben. Sie basieren auf den Python-Flashcards von David Whale, übersetzt und auf die Syntax von Python 3 umgebaut. Was es damit genau auf sich hat, finden Sie bei *bit.ly/pythonflashcards3*. Diese Flashcards erklären nicht die technischen Hintergründe, sondern beschreiben nur anhand ganz kurzer Beispiele die Syntax, also wie etwas gemacht wird.

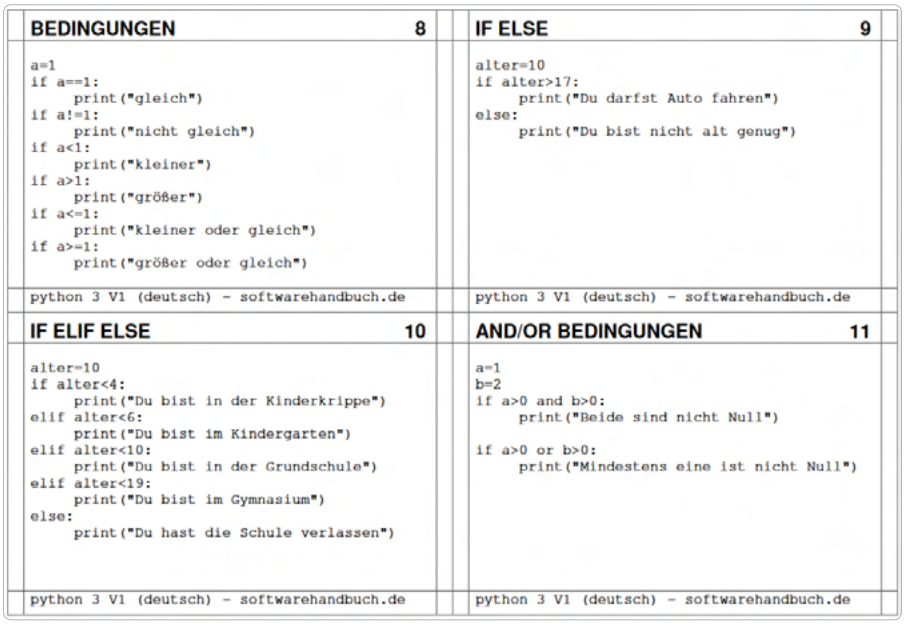


Bild 2.10: Ausschnitt aus den Python-Flashcards.

Ausgabe auf dem Bildschirm

Der Befehl `print` aus Python 2.x wird in Python 3.x durch die Funktion `print()` ersetzt. Hier zeigt sich der deutlichste Syntaxunterschied zwischen beiden Versionen.

```
print("Hallo Welt")

name = "Christian"
print(name)

print("Hallo " + name + " wie geht es Dir?")
```

Python – Variablen vom Typ Number

Number-Variablen enthalten Zahlenwerte, mit denen das Programm rechnen kann.

```
sekInMin = 60
minInStd = 60
stdInTag = 24
sekInTag = sekInMin * minInStd * stdInTag
print(sekInTag)
```

Python – Variablen vom Typ String

String-Variablen enthalten beliebige Zeichenketten und können mit dem +-Operator miteinander verknüpft werden.

```
vorname = "Fred"
nachname = "Schmidt"
name = vorname + " " + nachname
gruss = "Hallo "
gruss += name
print(gruss)
```

Python – Eingabe durch den Benutzer

Die Eingabefunktion `input()` ermöglicht Benutzereingaben. Diese werden in String-Variablen gespeichert. Um Zahlen einzugeben, müssen die String-Variablen in Zahlenwerte umgewandelt werden. Nach dem Fragezeichen in den Fragen steht immer noch ein Leerzeichen, damit die Eingabe nicht unmittelbar auf das Fragezeichen folgt, sondern, wie in einem deutschen Satz, durch ein Leerzeichen getrennt ist.

```
name = input("wie ist Dein Name? ")
print("Hallo " + name)

alter = int(input("Wie alt bist Du? "))
print("nächstes Jahr bist Du " + str(alter+1))
```

Python – Bedingungen mit if

Das Wort `if` (englisch: wenn) steht für eine Bedingung. Ist sie erfüllt, wird der folgende eingerückte Programmteil ausgeführt.

```
alter=10
if alter > 16:
   print("Du bist fertig mit der Schule")
a=1
if a==1:
   print("gleich")
if a!=1:
   print("nicht gleich")
if a<1:
   print("kleiner")
if a>1:
   print("größer")
```

```
if a<=1:
   print("kleiner oder gleich")
if a>=1:
   print("größer oder gleich")
```

EINRÜCKUNGEN SIND IN PYTHON WICHTIG

In den meisten Programmiersprachen werden Programmschleifen oder Entscheidungen eingerückt, um den Programmcode übersichtlicher zu machen. In Python dienen diese Einrückungen nicht nur der Übersichtlichkeit, sondern sind für die Programmlogik zwingend erforderlich. Dafür braucht man in dieser Sprache keine speziellen Satzzeichen, um Schleifen oder Entscheidungen zu beenden.

Python – Bedingungen mit if – else

Hinter dem Programmteil, der ausgeführt wird, wenn die Bedingung erfüllt ist, kann ein weiterer Block mit dem Schlüsselwort `else` stehen. Der darauf folgende Programmteil wird ausgeführt, wenn die Bedingung nicht erfüllt ist.

```
alter=10
if alter>17:
   print("Du darfst Auto fahren")
else:
   print("Du bist nicht alt genug")
```

Python – Bedingungen mit if – elif – else

Gibt es mehr Alternativen als nur richtig und falsch, lassen sich mit dem Wort `elif` weitere Bedingungen einfügen. Sie werden nur abgefragt, wenn keine der vorherigen Bedingungen wahr ist. Ist keine der Bedingungen wahr, wird der letzte Programmblock hinter `else` ausgeführt.

```
alter=10
if alter<4:
   print("Du bist in der Kinderkrippe")
elif alter<6:
   print("Du bist im Kindergarten")
elif alter<10:
   print("Du bist in der Grundschule")
elif alter<19:
   print("Du bist im Gymnasium")
```

```
else:
   print("Du hast die Schule verlassen")
```

Python – Bedingungen mit and und or verknüpfen

Mehrere Bedingungen lassen sich miteinander verknüpfen. Bei einer Verknüpfung mit `and` müssen alle einzelnen Bedingungen erfüllt sein, bei einer Verknüpfung mit `or` mindestens eine.

```
a=1
b=2
if a>0 and b>0:
   print("Beide sind nicht Null")

if a>0 or b>0:
   print("Mindestens eine ist nicht Null")
```

Python – Schleifen mit for

Schleifen mit `for` arbeiten eine bestimmte Anzahl von Durchläufen ab. Dabei kann auch ein Wertebereich oder eine Zeichenfolge angegeben werden. Die Schleife wird dann für jedes Zeichen der Zeichenfolge einmal ausgeführt.

```
total=20
for n in range(total):
   print(n)

for n in range(1,20):
   print(n)

name="Fred"
for ch in name:
   print(ch)
```

Python – Schleifen mit while

Schleifen mit `while` werden so lange ausgeführt, wie die Bedingung erfüllt ist.

```
print("Bohnen auf einem Schachbrett")
print("lege 1 Bohne auf das erste Feld")
print("lege 2 Bohnen auf das zweite Feld")
print("lege 4 Bohnen auf das dritte Feld")
```

```
print("wie lange, bis es 1000 Bohnen sind? ")
felder=0
bohnen=1
total=0
while total<1000:
   total += bohnen
   bohnen *= 2
   felder += 1
print("es dauert " + str(felder) + " Felder")
```

Die Zeilen, die mit # beginnen, sind Kommentare zur Verständlichkeit des Programms. Diese Zeilen werden vom Python-Interpreter nicht beachtet.

Python – Funktionen ohne Parameter

Soll ein bestimmter Programmteil mehrfach und von verschiedenen Stellen im Programm aufgerufen werden, definiert man eine Funktion, anstatt den Programmtext immer wieder zu kopieren.

```
def meinname():
   print("Mein Name ist Christian")

meinname()
Python - Funktionen mit Parametern
Definiert man mit einer Funktion einen oder mehrere Parameter, liefert
die Funktion je nach den übergebenen Parametern verschiedene Ergebnisse.
def zeigename(name):
   print("Mein Name ist " + name)

def info(name, alter):
   print("Mein Name ist " + name)
   print("Mein Alter ist " + str(alter))

zeigename("Fred")
zeigename("Harry")

info("Fred", 10)
info("Harry", 20)
```

Python – Funktionen mit Rückgabewert

Eine Funktion kann einen Wert zurückgeben, der mit `return` definiert wird. Der aufrufende Programmteil kann anschließend mit dem Rückgabewert der Funktion weiterrechnen.

```
def quadrat(n):
   return n*n

print(quadrat(5))
print(quadrat(10))

a=100
print(quadrat(a))
print(quadrat(a+10))

b=quadrat(a)
print(b)
```

Boolesche Wahr- und Falsch-Werte

Kann ein Wert nur den Zustand »Wahr« (`True`) oder »Falsch« (`False`) annehmen, kann dies als boolescher Wert gespeichert und ganz einfach abgefragt werden. Es ist kein Umweg über Zahlen, wie z. B. 1 oder 0, nötig.

```
nochmal = True
while nochmal:
    print("Hallo")

    antwort = input("Nochmal spielen?")
    if antwort != "Ja" and antwort != "ja":
        nochmal = False

    print("denke...")
    if nochmal:
        print("Noch eine Runde")
```

Wichtige Unterschiede zu Python 2

Python 3 ist an vielen Stellen nicht hundertprozentig kompatibel zu Python 2, weshalb Programme und auch Zusatzmodule immer nur mit einer der Versionen verwendet werden können. Hier die drei wichtigsten Unterschiede, über die man bei der Verwendung älterer Programme immer wieder stolpert:

Integerdivision

Python 3 dividiert zwei Ganzzahlen immer mathematisch korrekt, wobei als Ergebnis eine Fließkommazahl herauskommen kann. Python 2 dividiert zwei Ganzzahlen per Ganzzahldivision. Das Ergebnis ist auch immer ganzzahlig.

print()

Zur Ausgabe auf dem Bildschirm wird in Python 3 die Funktion `print()` verwendet. Die anzuzeigenden Werte stehen also immer in einer Klammer.

input()

Die Funktion `input()` liefert in Python 3 immer eine Zeichenkette und nicht wie in Python 2 einen Ganzzahlwert. Mit der Funktion `int()` kann eine eingegebene Zeichenfolge aus Ziffern in eine Ganzzahl umgerechnet werden.

99 Bottles of Beer

Immer nur »Hallo Welt« bei der Erfindung einer neuen Programmiersprache ist langweilig geworden. **99 Bottles of beer** heißt die neue Herausforderung für Entwickler von Programmiersprachen.

Dabei geht es darum, ein bekanntes englisches Trinklied, dessen Strophen sich weitgehend wiederholen, mit Hilfe eines Programms automatisiert auszugeben.

```
99 bottles of beer on the wall, 99 bottles of beer.
Take one down and pass it around, 98 bottles of beer on the wall.

98 bottles of beer on the wall, 98 bottles of beer.
Take one down and pass it around, 97 bottles of beer on the wall.

97 bottles of beer on the wall, 97 bottles of beer.
Take one down and pass it around, 96 bottles of beer on the wall.

...
```

```
3 bottles of beer on the wall, 3 bottles of beer.
Take one down and pass it around, 2 bottles of beer on the wall.

2 bottles of beer on the wall, 2 bottles of beer.
Take one down and pass it around, 1 bottle of beer on the wall.

1 bottle of beer on the wall, 1 bottle of beer.
Take one down and pass it around, no more on the wall.

No more bottles of beer on the wall, no more bottles of beer.
Go to the store and buy some more, 99 bottles of beer on the wall.
```

Interessant sind erst die letzten drei Strophen, wenn zunächst aus dem Plural `"bottles"` der Singular `"bottle"` und danach `"no more"` wird.

Die Webseite *www.99-bottles-of-beer.net* hat Programme in etwa 1.500 Programmiersprachen und Varianten gesammelt, die alle diesen Liedtext ausgeben.

Anstatt uns mit länger Programmiertheorie, Algorithmen und Datentypen aufzuhalten, schreiben wir gleich ein einfaches »99 Bottles of beer«-Programm.

Klicken Sie in Mu auf das Symbol *Neu*. Es öffnet sich ein neues Fenster, in das Sie den folgenden Programmcode eintippen:

```
#!/usr/bin/python
for zahl in range(99, 0, -1):
  if zahl > 1:
    print(zahl, "bottles of beer on the wall,", zahl, "bottles of beer.")
    if zahl > 2:
      bottles = str(zahl - 1) + " bottles of"
    else:
      bottles = "1 bottle of"
  else:
    print("1 bottle of beer on the wall, 1 bottle of beer.")
    bottles = "no more"
  print("Take one down, pass it around,", bottles, "beer on the wall.\n")
print("No more bottles of beer on the wall, no more bottles of beer.")
print("Go to the store and buy some more. 99 bottles of beer on the wall.")
```

Speichern Sie die Datei über das Symbol *Speichern* als `99-bottles-of-beer.py` ab. Oder Sie öffnen die fertige Programmdatei aus dem Download in der Python-Shell mit dem Symbol *Laden*. Die Farbcodierung im Quelltext erscheint automatisch und hilft dabei, Tippfehler zu finden.

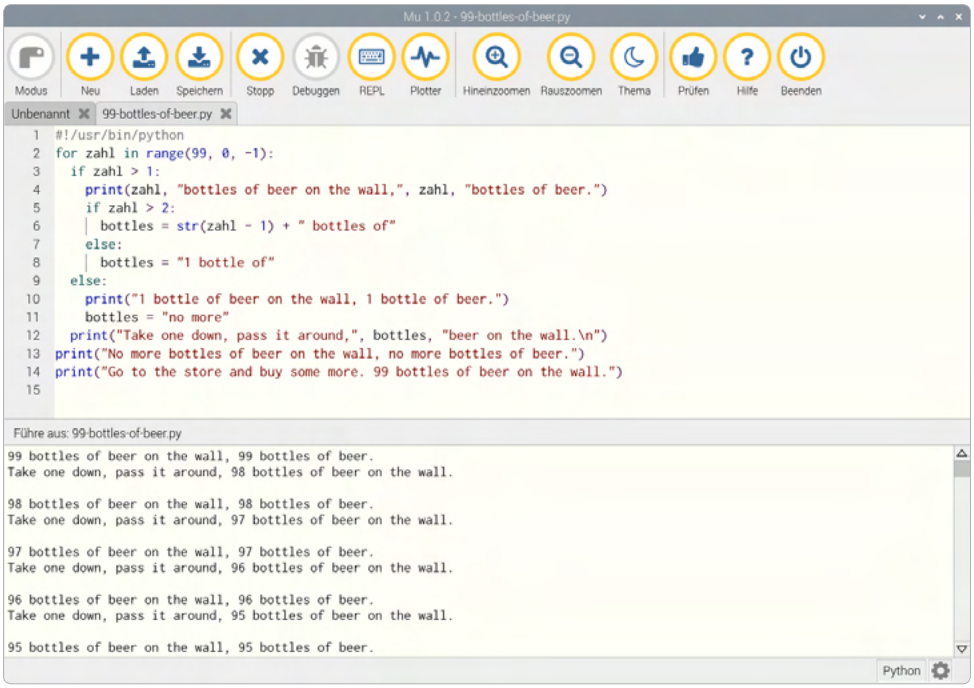


Bild 2.11: Das Programm 99-bottles-of-beer und die ersten Zeilen des Lieds in Mu.

Starten Sie das Programm mit dem Symbol *Ausführen*.

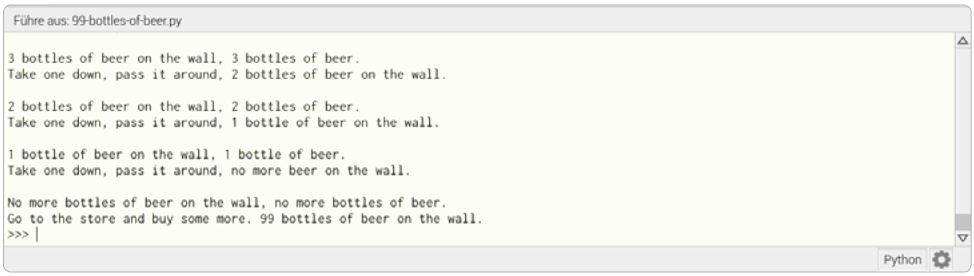


Bild 2.12: Die letzten Zeilen des Lieds.

So funktioniert es

Dass das Programm funktioniert, lässt sich einfach ausprobieren. Es stellen sich natürlich einige Fragen: Was passiert im Hintergrund? Was bedeuten die einzelnen Programmzeilen?

```
#!/usr/bin/python
```

Python-Programme, die über die Kommandozeile gestartet werden, müssen am Anfang immer obige Zeile enthalten. Bei Programmen, die nur über die Python-Shell gestartet werden, ist das nicht nötig. Aus Gründen der Kompatibilität sollten Sie sich aber angewöhnen, diese Zeile am Anfang jedes Python-Programms einzutragen.

So starten Sie das Programm in einem Kommandozeilenfenster:

```
python3 99-bottles-of-beer.py
```

Dabei wird kein Python-Shell-Fenster geöffnet, sondern der Liedtext direkt im Kommandozeilenfenster ausgegeben. Das funktioniert auch, wenn gar keine grafische Oberfläche auf dem Raspberry Pi verwendet wird. Das können Sie einfach ausprobieren, indem Sie mit der Tastenkombination Strg+Alt+F1 auf die Konsole wechseln und das Programm aufrufen. Mit der Tastenkombination Strg+Alt+F7 kommen Sie wieder zur grafischen Oberfläche zurück.

```
for zahl in range(99, 0, -1):
```

Die Hauptschleife des Programms ist eine for-Schleife, die 99-mal abläuft. Jede for-Schleife hat drei Parameter.

Der erste Parameter, hier `99`, gibt den Startwert des Schleifenzählers an. Dieser Schleifenzähler `zahl` gibt die Anzahl der noch verbliebenen Biere in der ersten der beiden Textzeilen einer Strophe an.

Der zweite Parameter, hier `0`, gibt den ersten Wert an, der nicht mehr erreicht wird. Die Schleife läuft nur, bis der Zähler `zahl` den Wert 1 hat.

Der dritte Parameter, hier `-1`, gibt an, um wie viel der Schleifenzähler in jedem Schleifendurchlauf verändert wird. Die hier verwendete Schleife zählt jedes Mal um 1 herunter.

```
  if zahl > 1:
    print(zahl, "bottles of beer on the wall,", zahl, "bottles of beer.")
```

Ist der Schleifenzähler größer als 1, also mindestens 2, wird die typische erste Zeile jeder Strophe ausgegeben, wobei die aktuelle Zahl an Bieren zweimal im Text vorkommt.

```
    if zahl > 2:
      bottles = str(zahl - 1) + " bottles of"
```

In diesem Fall wird jetzt noch geprüft, ob der Schleifenzähler auch größer als 2, also mindestens 3 ist. In diesem Fall wird die Zahl um 1 verringert und daraus ein Text erstellt, der später in die zweite Textzeile der Strophe eingebaut wird. Diese Zeichenfolge beginnt mit einer Zahl und endet auf `" bottles of"`. Sie wird in der Zeichenkettenvariable `bottles` gespeichert.

```
    else:
      bottles = "1 bottle of"
```

Ist der Schleifenzähler nicht mehr größer als 2, also genau 2, da er an dieser Stelle in der äußeren Abfrage nicht kleiner sein kann, wird die Zeichenkette `bottles` für die zweite Zeile auf `"1 bottle of"` gesetzt.

```
  else:
    print("1 bottle of beer on the wall, 1 bottle of beer.")
    bottles = "no more"
```

Hat im letzten Durchlauf der Schleife die Anzahl verbliebener Biere den Wert 1 erreicht, werden die Anweisungen hinter `else:` ausgeführt. Die erste Textzeile lautet jetzt anders, da das Wort `bottles` durch `bottle` ersetzt werden muss. Die Zeichenkette `bottles` für die zweite Zeile wird auf `"no more"` gesetzt.

```
   print("Take one down, pass it around,", bottles, "beer on the wall.\n")
```

In beiden Fällen wird die zweite Zeile der Strophe ausgegeben. Sie enthält, eingebettet in einen Standardtext, die Variable `bottles` mit der Zahl der übrigen Biere. Das Zeichen `"\n"` ganz am Ende fügt den zusätzlichen Zeilenumbruch für die Leerzeile zwischen zwei Strophen hinzu.

```
print("No more bottles of beer on the wall, no more bottles of beer.")
print("Go to the store and buy some more. 99 bottles of beer on the wall.")
```

Nach dem Ende der Schleife, wenn die Anzahl der verbliebenen Biere den Wert 0 erreicht hat, wird die letzte Strophe ausgegeben. Die beiden Anweisungen sind hier wieder komplett ausgerückt, da sie nicht mehr innerhalb der Schleife ausgeführt werden sollen.

Python-Programme brauchen keine eigene Anweisung zum Beenden. Sie enden einfach nach dem letzten Befehl bzw. nach einer Schleife, die nicht mehr ausgeführt wird und der keine weiteren Befehle folgen.

Zahlenraten mit Python

Das nächste Programm ist ein einfaches Ratespiel, in dem eine vom Computer zufällig gewählte Zahl vom Spieler in möglichst wenigen Schritten erraten werden soll. Sie finden das Programm als `spiel01.py` im Downloadarchiv.

```
#!/usr/bin/python
import random
zahl = random.randrange(0,1000)
```

```
tipp = 0
i = 0
print("Rate eine Zahl zwischen 0 und 1000")

while tipp != zahl:
  tipp = int(input("Dein Tipp:"))
  if zahl < tipp:
    print("Die gesuchte Zahl ist kleiner als",tipp)
  if zahl > tipp:
    print("Die gesuchte Zahl ist größer als",tipp)
  i += 1
print("Du hast die Zahl beim",i,". Tipp erraten")
```

Das Spiel verzichtet der Einfachheit halber auf jede grafische Oberfläche sowie auf erklärende Texte oder Plausibilitätsabfragen der Eingabe. Im Hintergrund generiert der Computer eine Zufallszahl zwischen 0 und 1.000. Geben Sie einfach einen Tipp ab, und Sie erfahren, ob die gesuchte Zahl größer oder kleiner ist. Mit weiteren Tipps tasten Sie sich an die richtige Zahl heran.

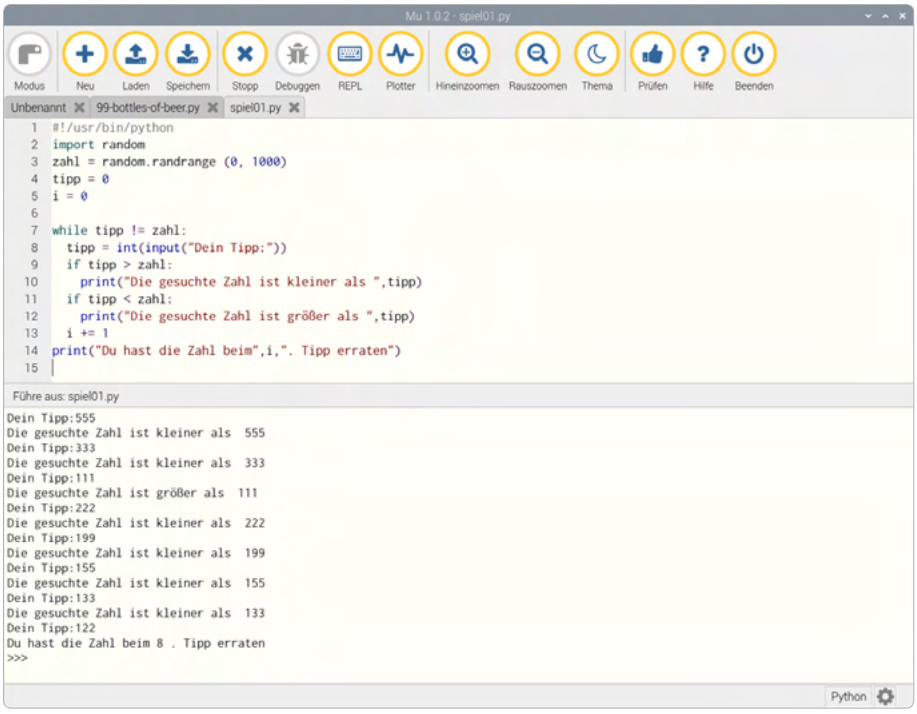


Bild 2.13: Zahlenraten in Python.

WIE ENTSTEHEN ZUFALLSZAHLEN?

Gemeinhin denkt man, in einem Programm könne nichts zufällig geschehen. Wie also kann ein Programm dann in der Lage sein, zufällige Zahlen zu generieren? Teilt man eine große Primzahl durch irgendeinen Wert, ergeben sich ab der x-ten Nachkommastelle Zahlen, die kaum noch vorhersehbar sind und sich auch ohne jede Regelmäßigkeit ändern, wenn man den Divisor regelmäßig erhöht. Dieses Ergebnis ist zwar scheinbar zufällig, lässt sich aber durch ein identisches Programm oder den mehrfachen Aufruf des gleichen Programms jederzeit reproduzieren. Nimmt man aber eine aus einigen dieser Ziffern zusammengebaute Zahl und teilt sie wiederum durch eine Zahl, die sich aus der aktuellen Uhrzeitsekunde oder dem Inhalt einer beliebigen Speicherstelle des Rechners ergibt, kommt ein Ergebnis heraus, das sich nicht reproduzieren lässt und daher als Zufallszahl bezeichnet wird.

So funktioniert es

```
import random
```

Um die zufällige Zahl zu generieren, wird ein externes Python-Modul namens `random` importiert, das diverse Funktionen für Zufallsgeneratoren enthält.

```
zahl = random.randrange(0,1000)
```

Die Funktion `randrange()` aus dem Modul `random` generiert eine Zufallszahl in dem durch die Parameter begrenzten Zahlenbereich, hier zwischen 0 und 999. Der Parameter der Funktion `random.randrange()` gibt die Anzahl möglicher Zufallszahlen an, mit 0 beginnend also immer die erste Zahl, die nicht erreicht wird. Das Gleiche gilt für Schleifen und ähnliche Funktionen in Python.

Diese Zufallszahl wird in der Variablen `zahl` gespeichert. Variablen sind in Python Speicherplätze, die einen beliebigen Namen haben und Zahlen, Zeichenfolgen, Listen oder andere Datenarten speichern können. Anders als in einigen anderen Programmiersprachen müssen sie nicht vorher deklariert werden.

```
tipp = 0
```

Die Variable `tipp` enthält später die Zahl, die der Benutzer tippt. Am Anfang ist sie `0`.

```
i = 0
```

Die Variable `i` hat sich unter Programmierern als Zähler für Programmschleifendurchläufe eingebürgert. Hier wird sie verwendet, um die Anzahl der Tipps zu zählen, die der Benutzer brauchte, um die geheime Zahl zu erraten. Auch diese Variable steht am Anfang auf `0`.

```
print("Rate eine Zahl zwischen 0 und 1000")
```

Damit der Benutzer weiß, was er tun muss, wird eine kurze Textzeile angezeigt, bevor die Hauptschleife des Spiels startet.

```
while tipp != zahl:
```

Das Wort `while` (englisch für »so lange wie«) leitet eine Programmschleife ein, die in diesem Fall so lange wiederholt wird, wie `tipp`, die Zahl, die der Benutzer tippt, ungleich der geheimen Zahl `zahl` ist. Python verwendet die Zeichenkombination `!=` für ungleich. Hinter dem Doppelpunkt folgt die eigentliche Programmschleife.

```
    tipp = int(input("Dein Tipp:"))
```

Die Funktion `input()` schreibt den Text `Dein Tipp:` und erwartet danach eine Eingabe, die eine beliebige Zeichenkette sein kann. Die Funktion `int()` ermittelt daraus eine Ganzzahl, die in der Variablen `tipp` gespeichert wird. Diese Zeile zeigt, wie mehrere Funktionen ineinander geschachtelt werden können.

```
    if zahl < tipp:
```

Wenn die geheime Zahl `zahl` kleiner ist als die vom Benutzer getippte Zahl `tipp`, wird dieser Text ausgegeben:

```
        print("Die gesuchte Zahl ist kleiner als",tipp)
```

Am Ende steht die Variable `tipp`, damit die getippte Zahl im Text angezeigt wird. Trifft diese Bedingung nicht zu, wird die eingerückte Zeile einfach übergangen.

```
    if tipp < zahl:
```

Wenn die geheime Zahl `zahl` größer ist als die vom Benutzer getippte Zahl `tipp`, wird ein anderer Text ausgegeben:

```
        print("Die gesuchte Zahl ist größer als",tipp)
    i += 1
```

In jedem Fall – deshalb nicht mehr eingerückt – wird der Zähler `i`, der die Versuche zählt, um 1 erhöht. Diese Zeile mit dem Operator `+=` bedeutet das Gleiche wie `i = i + 1`.

```
print("Du hast die Zahl beim",i,". Tipp erraten")
```

Diese Zeile ist nicht mehr eingerückt, was bedeutet, dass auch die `while`-Schleife zu Ende ist. Trifft deren Bedingung nicht mehr zu, ist also die vom Benutzer getippte Zahl `tipp` nicht mehr ungleich (sondern gleich) der geheimen Zahl `zahl`, wird dieser Text ausgegeben, der sich aus zwei Satzteilen und der Variablen `i` zusammensetzt und angibt, wie viele Versuche der Benutzer benötigte.

3 GRAFISCHE BENUTZEROBERFLÄCHEN FÜR PYTHON-PROGRAMME

Kein modernes Programm, das irgendeine Interaktion mit dem Benutzer erfordert, läuft heute mehr im reinen Textmodus. Überall gibt es grafische Oberflächen, auf denen man Buttons anklicken kann, anstatt Eingaben über die Tastatur vornehmen zu müssen.

Python selbst bietet keine grafischen Oberflächen für Programme, es gibt aber mehrere externe Module, die speziell dafür da sind, grafische Oberflächen zu erstellen. Eines der bekanntesten derartigen Module ist **tkinter**, das die grafische Oberfläche Tk, die auch für diverse andere Programmiersprachen genutzt werden kann, für Python verfügbar macht.

Die Strukturen des grafischen Toolkits Tk unterscheiden sich etwas von Python und mögen auf den ersten Blick ungewöhnlich erscheinen. Deshalb fangen wir mit einem ganz einfachen Beispiel an.

Rechner für britische Maßeinheiten

Die historisch überlieferten britischen Maßeinheiten sind in einigen Bereichen des Alltags zumindest für Urlauber noch relevant. Da sich kaum jemand die Umrechnungsformeln auswendig merken kann, rechnet ein einfaches Programm eine Temperaturangabe in Grad Fahrenheit in die hierzulande üblichen Grad Celsius um.

Das Programm `fahrenheit.py` zeigt die Grundfunktionen der Tkinter-Bibliothek zum Aufbau grafischer Dialogfelder. Dabei ergibt sich die Größe der Dialogfelder und Steuerelemente aus der jeweils erforderlichen Größe automatisch, kann aber bei Bedarf nachträglich manuell beeinflusst werden.

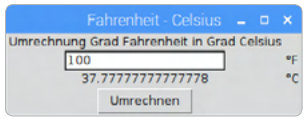


Bild 3.1: So wird das fertige Dialogfeld aussehen.

```
#!/usr/bin/python
from tkinter import *
root = Tk()
root.title("Fahrenheit - Celsius")
gradc = DoubleVar()

def finc():
  c = 5 / 9 * (int(f.get()) - 32)
  gradc.set(c)
```

```
Label(root, text="Umrechnung Grad Fahrenheit in Grad Celsius").grid(row=0)
f = Entry(root)
f.grid(row=1, column=0)
Label(root, text="°F").grid(row=1, column=1)
Label(root, textvariable=gradc).grid(row=2, column=0)
Label(root, text="°C").grid(row=2, column=1)
Button(root, text="Umrechnen", command=finc).grid(row=3)
root.mainloop()
```

So funktioniert es

```
from tkinter import *
```

In jedem Programm, das TKinter für grafische Benutzeroberflächen nutzt, müssen die Elemente der Tkinter-Bibliothek importiert werden.

```
root = Tk()
```

Tkinter arbeitet mit sogenannten Widgets. Dabei handelt es sich um eigenständige Bildschirmelemente, in den meisten Fällen um Dialogfelder, die ihrerseits verschiedene Elemente enthalten. Jedes Programm braucht ein `root`-Widget, von dem aus alle weiteren Objekte aufgerufen werden. Dieses `root`-Widget heißt immer `Tk()`, generiert automatisch ein Fenster und initialisiert auch die Tkinter-Bibliothek.

```
root.title("Fahrenheit - Celsius")
```

Objekte in Tkinter stellen verschiedene Methoden für unterschiedliche Zwecke zur Verfügung. Die Methode `title()` in einem Widget setzt den Fenstertitel, schreibt also in diesem Fall die Wörter `Fahrenheit - Celsius` in die Titelzeile des neuen Fensters.

Jedes Widget kann mehrere Objekte enthalten, die einzeln definiert werden. Tkinter kennt dazu verschiedene Objekttypen, von denen jeder unterschiedliche Parameter ermöglicht, die die Eigenschaften des Objekts beschreiben. Die Parameter werden, durch Kommata getrennt, in einer Klammer hinter dem Objekttyp angegeben.

```
gradc = DoubleVar()
```

Werte, die in Dialogfeldern dargestellt werden sollen, müssen in speziellen Tkinter-Variablen gespeichert werden. Nur auf diese Variablen können die Tkinter-Objekte direkt zugreifen. Eine Tkinter-Variable `gradc` enthält später das errechnete Ergebnis in Grad Celsius als Zeichenkette. Tkinter unterscheidet zwischen Integervariablen und

Variablen mit doppelter Genauigkeit, die auch zum Speichern von Zeichenketten verwendet werden.

```
def finc():
  c = 5 / 9 * (int(f.get()) - 32)
  gradc.set(c)
```

Die Funktion `finc()` rechnet °F in °C um. Solche Funktionen müssen einmal definiert werden und können dann später im Programm jederzeit verwendet werden.

Die Umrechnungsformel für °F in °C lautet:

c = 5/9 * (f – 32)

Für dieses Programm erweitern wir diese Formel noch etwas.

`f.get()` liest den vom Benutzer im Formular eingegebenen Wert in °F aus. Das Eingabefeld `f` wird später noch im Hauptprogramm definiert.

`int(f.get())` ermittelt einen ganzzahligen Zahlenwert aus der Eingabe. Eingabefelder in Tkinter liefern immer Zeichenketten. Zur Umrechnung wird aber ein Zahlenwert benötigt.

Das Ergebnis der Berechnung wird in der Variablen `c` gespeichert. Dabei handelt es sich um eine klassische Python-Variable, die nur innerhalb der Funktion `finc()` gilt. Um den Wert im Dialogfeld anzuzeigen, wird dieser Wert in der Tkinter-Variable `gradc` gespeichert.

Tkinter verwendet nicht das Gleichheitszeichen aus Python für die Zuweisung von Variablen, sondern die Methode `set()`. Solche Methoden werden in Python auch an anderen Stellen verwendet. Methoden werden mit einem Punkt getrennt an den Namen des jeweiligen Objektes oder der Variable angehängt.

Jetzt beginnt das Hauptprogramm, das die einzelnen Objekte des Dialogfelds und ihre Funktionen definiert.

```
Label(root,
      text="Umrechnung Grad Fahrenheit in Grad Celsius").grid(row=0)
```

Objekte vom Typ `Label` sind reine Texte in einem Widget. Diese können vom Programm verändert werden, bieten aber keine Interaktion mit dem Benutzer.

Der erste Parameter in jedem Tkinter-Objekt ist der Name des übergeordneten Widgets, meistens des Fensters, in dem sich das jeweilige Objekt befindet. In unserem Fall ist das das einzige Fenster im Programm, das `root`-Widget.

Der Parameter `text` enthält den Text, der auf dem Label angezeigt werden soll.

An das Ende der Objektdefinition wird der sogenannte Geometrie-Manager als Methode angehängt. Dieser Geometrie-Manager baut das Objekt in die Dialogbox ein und generiert die Geometrie des Widgets.

GEOMETRIE-MANAGER IN TKINTER

Tkinter bietet drei verschiedene Geometrie-Manager zum Gestalten von Dialogfeldern, wobei in jedem Dialogfeld immer nur einer für alle Objekte verwendet werden kann.

- `pack()` ist der einfachste Geometrie-Manager, bietet aber die wenigsten Gestaltungsmöglichkeiten. Es ist nicht immer ganz einfach, das Ergebnis von `pack()` vorherzusehen. Da nur wenige weitere Parameter benötigt werden, eignet sich `pack()` besonders zur schnellen Erstellung sehr einfacher Dialogfelder.
- `grid()` bietet wesentlich mehr Möglichkeiten zur Gestaltung von Dialogfeldern. Hier werden alle Objekte in Zeilen und Spalten angeordnet. Die tatsächliche Größe ergibt sich aber wie bei `pack()` aus der zur Darstellung der Werte erforderlichen Größe. Wegen seiner Übersichtlichkeit und großen Flexibilität wird in den meisten Fällen `grid()` verwendet.
- `place()` positioniert jedes Objekt über absolute oder relative Koordinaten. Hier ist wesentlich mehr Aufwand erforderlich, da die Größe jedes Objekts genau festgelegt werden muss. Nichts wird mehr automatisch ermittelt.

Dieses Programm verwendet den Geometrie-Manager `grid()`. Das Label liegt in der ersten Zeile `(row=0)` des Dialogfelds. Da keine Spalte angegeben ist, wird das Label automatisch linksbündig angeordnet.

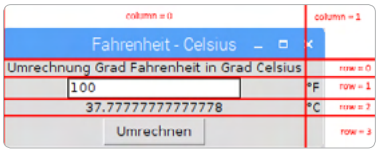


Bild 3.2: Spalten und Zeilen im Dialogfeld.

Auf die gleiche Weise werden die weiteren Objekte im Dialogfeld angeordnet.

```
f = Entry(root)
f.grid(row=1, column=0)
```

Das nächste Objekt ist ein Eingabefeld für den Wert in °F, vom Typ `Entry()`. Es wird der Variable `f` zugewiesen, um es über die zuvor definierte Funktion mit der Methode `f.get()` auslesen zu können.

Mit dem Geometrie-Manager `grid()` wird dieses Objekt in der ersten Spalte und zweiten Zeile angeordnet.

```
Label(root, text="°F").grid(row=1, column=1)
```

Das Label, das den Text °F rechts neben dem Eingabefeld anzeigt, wird in der zweiten Spalte der gleichen Zeile angeordnet.

```
Label(root, textvariable=gradc).grid(row=2, column=0)
```

Das errechnete Ergebnis in °C wird ebenfalls in einem Label angezeigt. Statt des Parameters `text` für einen statischen Text wird hier der Parameter `textvariable` verwendet. Dieser zeigt immer den aktuellen Inhalt der Tkinter-Variable `gradc` an. Die weiter oben definierte Funktion `finc()` schreibt das Ergebnis in diese Variable. Es wird vom Label automatisch übernommen.

```
Label(root, text="°C").grid(row=2, column=1)
```

Das Label, das den Text °C rechts neben dem Label mit dem Ergebnis anzeigt, wird in der zweiten Spalte der gleichen Zeile angeordnet.

```
Button(root, text="Umrechnen", command=finc).grid(row=3)
```

Objekte vom Typ `Button` sind Schaltflächen, die der Benutzer anklickt, um eine bestimmte Aktion auszulösen. Auch hier enthält der Parameter `text` den Text, der auf dem Button angezeigt werden soll.

Der Parameter `command` enthält eine Funktion, die der Button beim Anklicken aufruft. Dabei können keine Parameter übergeben werden, und der Funktionsname muss ohne Klammern angegeben werden. Dieser Button ruft die Funktion `finc()` auf, die den im Eingabefeld eingegebenen Wert in °C umrechnet und in der Tkinter-Variable `gradc` speichert.

```
root.mainloop()
```

Das Hauptprogramm besteht nur aus einer einzigen Zeile. Es startet die Hauptschleife `mainloop()`, eine Methode des `root`-Widgets. Diese Programmschleife wartet darauf, dass der Benutzer eines der Widgets betätigt und damit eine Aktion auslöst. In diesem einfachen Programmbeispiel kann der Benutzer nur im Eingabefeld einen Wert eingeben und anschließend auf den Button klicken.

Das x-Symbol oben rechts zum Schließen des Fensters braucht bei Tkinter nicht eigens definiert zu werden. Schließt der Benutzer das Hauptfenster `root`, wird automatisch die Hauptschleife `mainloop()` beendet.

Zahlenraten mit grafischer Oberfläche

Das Programm `spiel02.py` liefert eine grafische Oberfläche für das Zahlenratespiel aus einem früheren Beispiel. Das Spiel wird so optisch ansprechender und vor allem komfortabler zu bedienen. Außerdem werden Fehleingaben weitgehend vermieden, da der Spieler nur noch auf Buttons klickt und keine Werte mehr selbst eingibt.

Bild 3.3: So sieht das fertige Spiel aus.

Das Programm zeigt einige weitere Parameter und Programmiertipps in Tkinter.

```
#!/usr/bin/python
from tkinter import *
import random

root = Tk()
root.title("Zahlenraten")
zahl = random.randrange (0, 1000)
tipp = 500
z = IntVar()
z.set(tipp)
i = 0
msg1 = DoubleVar()
msg1.set(" ")
msg2 = DoubleVar()
msg2.set(" ")
msg3 = DoubleVar()
msg3.set("Tipps: 0")

def minus10():
  global tipp
  tipp -= 10
  if tipp<0:
    tipp = 0
  z.set(tipp)

def minus1():
  global tipp
  tipp -= 1
  if tipp<0:
    tipp = 0
  z.set(tipp)
```

```
def plus1():
  global tipp
  tipp += 1
  if tipp<0:
    tipp = 0
  z.set(tipp)

def plus10():
  global tipp
  tipp += 10
  if tipp<0:
    tipp = 0
  z.set(tipp)

def tippen():
  global tipp
  global i
  if zahl < tipp:
    msg1.set("Die gesuchte Zahl ist kleiner als " + str(tipp))
  if zahl > tipp:
    msg1.set("Die gesuchte Zahl ist größer als " + str(tipp))
  if zahl == tipp:
    msg1.set("Die gesuchte Zahl ist gleich " + str(tipp))
    msg2.set("Erraten!")
  i += 1
  msg3.set("Tipps: " + str(i))

Label(root,
      text="Rate die geheime Zahl (0-1000)",
      width=32).grid(row=0, column=0, columnspan=5)
Label(root,
      textvariable=z,
      font="Verdana 24 bold",
      fg="blue").grid(row=1, column=2)
Button(root, text="-10", command=minus10).grid(row=2, column=0)
Button(root, text="-1", command=minus1).grid(row=2, column=1)
Button(root, text="Tipp", command=tippen).grid(row=2, column=2)
Button(root, text="+1", command=plus1).grid(row=2, column=3)
Button(root, text="+10", command=plus10).grid(row=2, column=4)
```

```
Label(root, textvariable=msg1).grid(row=3, column=0, columnspan=5)
Label(root,
      textvariable=msg2,
      font="Verdana 24 bold",
      fg="red").grid(row=4, column=0, columnspan=5)
Label(root, textvariable=msg3).grid(row=5, column=0, columnspan=5)
root.mainloop()
```

In den ersten Zeilen wird wie bereits bekannt, das Tkinter-Modul importiert und das root-Widget eingerichtet.

```
zahl = random.randrange (0, 1000)
tipp = 500
z = IntVar()
z.set(tipp)
i = 0
```

Wie in der früheren Textversion des Spiels wird eine zufällig ermittelte geheime Zahl in der Variablen `zahl` gespeichert. Die Variable `tipp` wird auf einen Startwert von 500 in der Mitte des möglichen Zahlenbereichs gesetzt. Bei grafischen Oberflächen sollten Sie generell darauf achten, alle einstellbaren Werte sinnvoll vorzubelegen.

Die Tkinter-Variable `z` soll später die aktuell eingestellte Zahl im Dialogfeld groß anzeigen. Diese Variable wird als Ganzzahlvariable vom Typ `IntVar()` definiert und anschließend auf den Wert der Python-Variable `tipp` gesetzt.

Die Variable `i`, die im Spiel die abgegebenen Tipps zählt, wird wie in der ersten Version des Spiels am Anfang auf 0 gesetzt.

```
msg1 = DoubleVar()
msg1.set(" ")
msg2 = DoubleVar()
msg2.set(" ")
msg3 = DoubleVar()
msg3.set("Tipps: 0")
```

Für die drei Textzeilen unter der Buttonleiste werden drei Tkinter-Variablen vom Typ `DoubleVar()` definiert. Die ersten beiden Zeilen sind am Anfang leer, die dritte Zeile zeigt, dass bis jetzt 0 Tipps abgegeben wurden.

Da der Spieler in dieser Version des Spiels keine Zahlenwerte eintippt, werden vier Funktionen definiert, mit denen sich der aktuelle Tipp um 1 oder 10 erhöhen und verringern lässt. Diese Funktionen werden später den Buttons zugewiesen, mit denen man seinen Tipp einstellt.

```
def minus10():
  global tipp
  tipp -= 10
  if tipp<0:
    tipp = 0
  z.set(tipp)
```

Diese Funktion verringert die eingestellte Zahl um 10. Damit die Variable `tipp`, die global im ganzen Spiel gilt, in der Funktion geschrieben und nicht nur gelesen werden kann, wird sie in der Funktion als `global` definiert.

Anschließend wird diese Variable um 10 verringert. Sollte `tipp` dabei einen Wert kleiner als 0 erreichen, wird sie automatisch auf 0 gesetzt. Damit werden ungültige Werte vermieden, selbst wenn der Spieler immer wieder auf den Button *-10* klickt.

Zum Schluss wird die Tkinter-Variable `z` auf den Wert der Python-Variable `tipp` gesetzt, um diesen Wert auf dem Label anzuzeigen.

Die anderen drei Funktionen zum Ändern der getippten Zahl sind ähnlich aufgebaut.

```
def tippen():
  global tipp
  global i
  if zahl < tipp:
    msg1.set("Die gesuchte Zahl ist kleiner als " + str(tipp))
  if zahl > tipp:
    msg1.set("Die gesuchte Zahl ist größer als " + str(tipp))
  if zahl == tipp:
    msg1.set("Die gesuchte Zahl ist gleich " + str(tipp))
    msg2.set("Erraten!")
  i += 1
  msg3.set("Tipps: " + str(i))
```

Die Funktion `tippen()` gibt den Tipp ab und vergleicht die getippte Zahl mit der geheimen Zahl. Dazu werden wieder die beiden verwendeten Variablen als global definiert.

Jetzt gibt es drei Möglichkeiten:

- Die gesuchte Zahl ist kleiner als der Tipp. In diesem Fall wird ein entsprechender Text in der Tkinter-Variablen `msg1` abgelegt, der dann auf dem Label angezeigt wird.
- Die gesuchte Zahl ist größer als der Tipp. Dieser Fall funktioniert im Prinzip genauso wie der erste.
- Die gesuchte Zahl ist gleich wie die des Tipps. In diesem Fall hat der Spieler die geheime Zahl erraten. Zusätzlich zum Text in der Tkinter-Variablen `msg1` wird in der bisher leeren Tkinter-Variablen `msg2` der Text `Erraten!` abgelegt.

In allen drei Fällen wird der Zähler `i`, der die abgegebenen Tipps zählt, um 1 erhöht und anschließend in der Tkinter-Variablen `msg3` eine Zeichenkette zusammengebaut, die die Anzahl der Tipps anzeigt.

GLEICH IST NICHT GLEICH

Python verwendet zwei Arten von Gleichheitszeichen. Das einfache = dient dazu, einer Variablen einen bestimmten Wert zuzuweisen. Das doppelte Gleichheitszeichen == wird in Abfragen verwendet und prüft, ob zwei Werte wirklich gleich sind.

Jetzt beginnt das Hauptprogramm mit den Definitionen der einzelnen Widgets für das Dialogfeld.

```
Label(root,
      text="Rate die geheime Zahl (0-1000)",
      width=32).grid(row=0, column=0, columnspan=5)
```

Das erste Label zeigt einen statischen Hinweistext an, um dem Spieler zu zeigen, was er tun soll. Dieses Label enthält einen zusätzlichen Parameter `width`, der die Breite des Widgets angibt. Indem Sie dem ersten Label die größtmögliche Breite fest vorgeben, verhindern Sie, dass bei Veränderungen der Texte das ganze Dialogfeld seine Breite verändert, was zwar funktioniert, aber nicht gut aussieht.

Wenn die Liste der Parameter in einem Label länger ist, schreibt man bei größeren Objekten jeden Parameter in eine eigene Zeile, sodass alle Parameter untereinander ausgerichtet sind. Im Gegensatz zu den Einrückungen bei Schleifen und Abfragen in Python sind diese Einrückungen der Tkinter-Objekte jedoch nicht obligatorisch.

Das Label wird in der ersten Spalte und ersten Zeile des Dialogfelds angezeigt. Der zusätzliche Parameter `columnspan=5` legt fest, dass sich dieses Label über alle fünf Spalten des Dialogfelds erstreckt. Andernfalls würde die erste Spalte so breit wie dieses Label, was die Zeile mit den Buttons auseinanderreißen würde.

```
Label(root,
      textvariable=z,
      font="Verdana 24 bold",
      fg="blue").grid(row=1, column=2)
```

Das nächste Label zeigt die vom Spieler eingestellte Zahl an. Sie wird automatisch aus der Tkinter-Variable `z` übernommen. Das Label verwendet nicht die Standardschriftart, sondern eine eigene, die über den Parameter `font` festgelegt wird. Der zusätzliche Parameter `fg` legt die Schriftfarbe fest. Dieses Label liegt in der zweiten Zeile in der dritten Spalte und ist nur diese eine Spalte breit.

```
Button(root, text="-10", command=minus10).grid(row=2, column=0)
Button(root, text="-1", command=minus1).grid(row=2, column=1)
Button(root, text="Tipp", command=tippen).grid(row=2, column=2)
Button(root, text="+1", command=plus1).grid(row=2, column=3)
Button(root, text="+10", command=plus10).grid(row=2, column=4)
```

Jetzt folgen die fünf Buttons, die alle eine der zuvor definierten Funktionen aufrufen. Jeder Button ist eine Spalte breit, sie liegen alle nebeneinander in der dritten Zeile des Dialogfelds.

```
Label(root,
      textvariable=msg1).grid(row=3, column=0, columnspan=5)
```

Das Label in der vierten Zeile des Dialogfelds zeigt den Inhalt der Tkinter-Variable `msg1`. Die Funktion `tippen()` schreibt jedes Mal, wenn ein Tipp abgegeben wurde, in diese Zeichenkette, ob die gesuchte Zahl kleiner oder größer als der Tipp ist.

```
Label(root,
      textvariable=msg2,
      font="Verdana 24 bold",
      fg="red").grid(row=4, column=0, columnspan=5)
```

Das Label in der vierten Zeile des Dialogfelds zeigt den Inhalt der Tkinter-Variable `msg2`. Dieses Label verwendet eine große rote Schriftart und erstreckt sich über alle fünf Spalten. Die Funktion `tippen()` schreibt, wenn der Spieler die Zahl erraten hat, den Text `Erraten!` in diese Variable.

```
Label(root,
      textvariable=msg3).grid(row=5, column=0, columnspan=5)
```

Das letzte Label zeigt den Inhalt der Tkinter-Variable `msg3`. Die Funktion `tippen()` schreibt jedes Mal, wenn ein Tipp abgegeben wurde, die Anzahl abgegebener Tipps in diese Zeichenkette.

```
root.mainloop()
```

Nachdem alle Widgets definiert sind, startet die Hauptschleife des Programms. Das Spiel kann beginnen.

Symbole

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z

MACH'S EINFACH

Erste Schritte

RASPBERRY PI PROGRAMMIEREN

Der perfekte Einstieg in die Programmierung mit Scratch und Python

Dieser Ratgeber macht den Einstieg in die Raspberry-Pi-Programmierung mit Scratch und Python extrem einfach, egal, ob Sie ein älteres Modell oder den neuen leistungsstarken Raspberry Pi 4 einsetzen. Das Buch verzichtet auf trockene Programmiertheorie und setzt auf anschauliche Beispielprogramme und überschaubare Projektbeispiele, anhand derer Sie die grundlegenden Techniken der Programmierung mit Scratch und Python schnell erlernen.

Los geht es mit einfachen Scratch-Projekten. Sie programmieren Ihr erstes Spiel und bauen Ihr eigenes Labyrinth.

Mit der auf dem Raspberry Pi eingebauten GPIO-Schnittstelle kann man auch direkt angeschlossene Elektronik über eigene Programme ansteuern, was auf einem PC nur mit erheblichem Aufwand möglich ist. Das geht am besten mit Python. Lassen Sie LEDs mit Python blinken, programmieren Sie einen Spielwürfel mit LEDs oder steuern Sie Punktmatrix-Anzeigen. Raspberry-Pi-Enthusiast Christian Immler nimmt Sie an die Hand und macht Sie fit für die Umsetzung unterschiedlichster Elektronikprojekte mit Scratch und Python auf dem Raspberry Pi.

IN DIESEM BUCH GEHT ES UM:

- Den Raspberry Pi vorbereiten
- Betriebssysteminstallation step-by-step
- Einfache Programme mit Scratch
- **KI:** Labyrinthe bauen und lösen
- Spiele mit Scratch programmieren
- Fußgängerampel mit Scratch steuern
- Alle Scratch-Blöcke im Überblick
- Einstieg in die Python-Programmierung
- **MU:** der neue Code-Editor
- Python-Flashcards
- Spielwürfel mit PyGame programmieren
- Zahlenraten mit grafischer Oberfläche
- Elektronik über GPIO steuern
- SenseHAT mit Python programmieren
- LED per Pulsweitenmodulation dimmen
- Siebensegmentanzeigen mit Python steuern
- Digitaluhr automatisch starten
- LCD-Module mit Python ansteuern
- So funktioniert das Pi-Camera-Modul
- und mehr

FRANZIS

Besuchen Sie unsere Webseite:
www.franzis.de