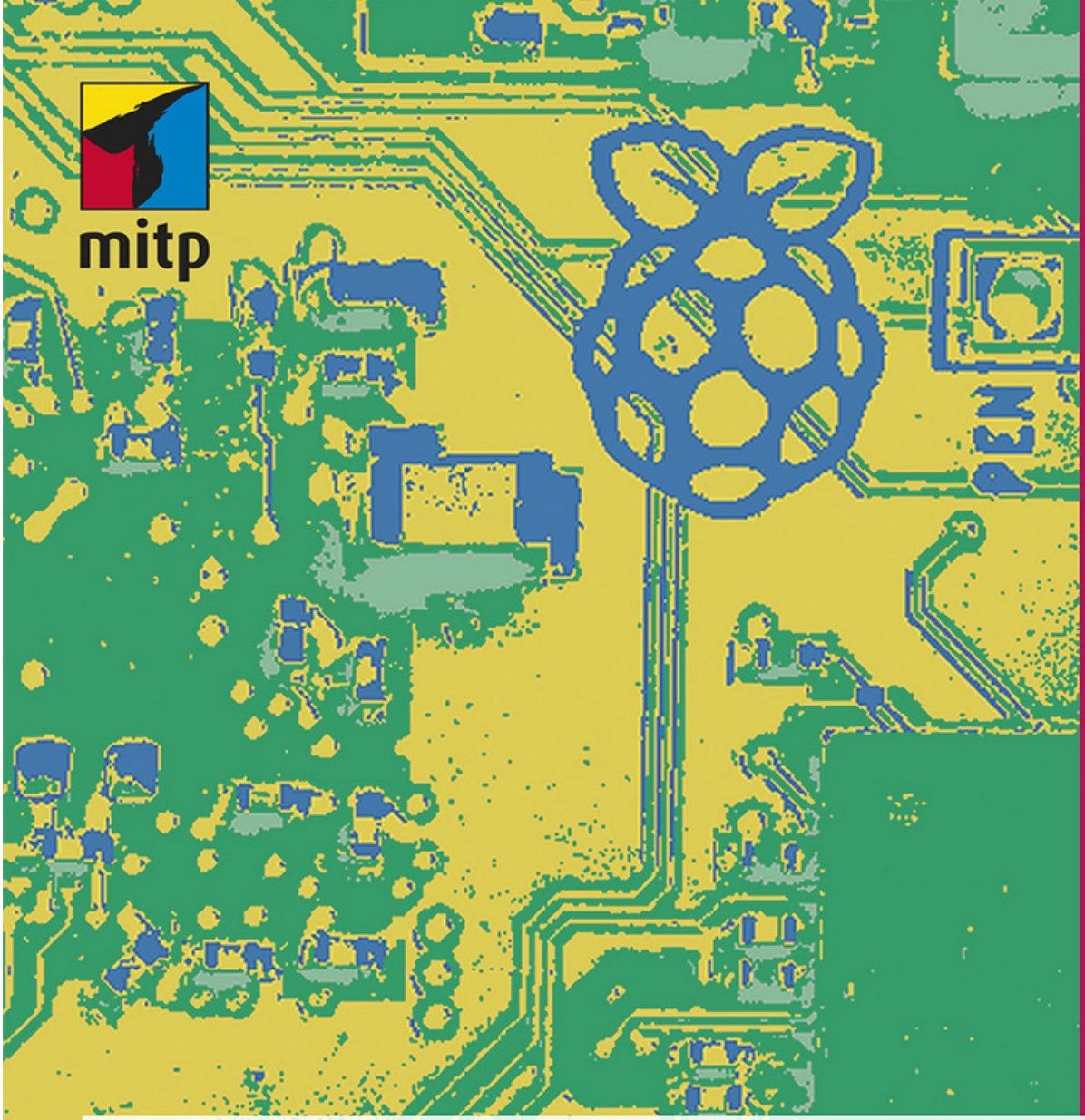

mitp

Michael Weigend

4. Auflage

Raspberry Pi programmieren mit Python

den Variablennamen eingeben:

```
>>> a
1257002
```

1.3.2 Hotkeys

Um effizient mit der Python-Shell arbeiten zu können, sollten Sie einige Tastenkombinationen (Hotkeys) beherrschen.

Manchmal möchten Sie ein früheres Kommando ein zweites Mal verwenden – vielleicht mit kleinen Abänderungen. Dann benutzen Sie die Tastenkombination Alt+P. Beispiel:

```
>>> 1 + 2*3 + 4
11
>>>
```

Wenn Sie jetzt *einmal* die Tastenkombination Alt+P betätigen, erscheint hinter dem letzten Prompt wieder das vorige Kommando (*previous*):

```
>>> 1 + 2*3 + 4
```

Wenn Sie nochmals diesen Hotkey drücken, verschwindet der Term wieder und es erscheint das *vorletzte* Kommando, beim nächsten Mal das vorvorletzte und so weiter. Die Shell merkt sich alle Ihre Eingaben in einer Folge, die man *History* nennt. Mit Alt+P und Alt+N können Sie in der History rückwärts- und vorwärtsgehen (Abbildung 1.2).

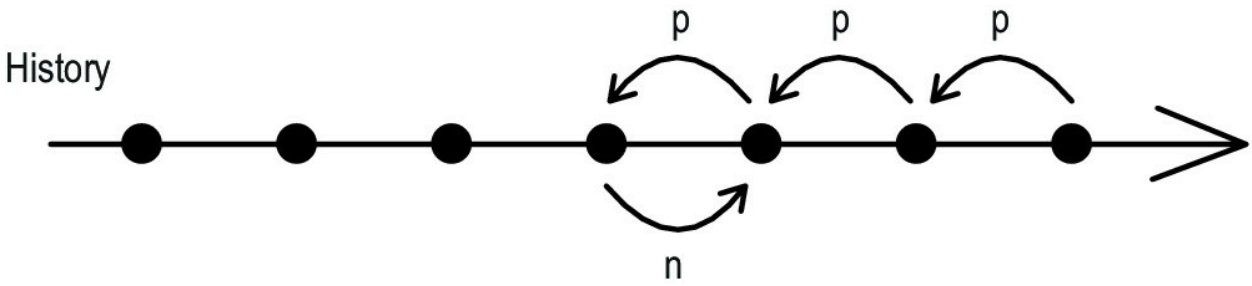

Abb. 1.2: Navigieren in der History mit Alt+P und Alt+N

Eine dritte wichtige Tastenkombination, die Sie sich merken sollten, ist Strg+C. Damit können Sie die Ausführung des gerade laufenden Programms abbrechen, z.B. wenn sie zu lange dauert.

Tastenkombination	Wirkung
Alt+P	Previous Command. Die vorige Anweisung in der History (Liste der bisherigen Anweisungen) erscheint hinter dem Prompt.
Alt+N	Next Command. Die nachfolgende Anweisung in der History erscheint hinter dem Prompt.
Strg+C	Keyboard Interrupt. Der Abbruch eines Programms (z.B. bei einer Endlosschleife) wird erzwungen.

Tabelle 1.1: Wichtige Tastenkombinationen der Python-Shell

1.4 Die Python-Shell als Taschenrechner

Die Python-Shell können Sie wie einen mächtigen und komfortablen Taschenrechner benutzen. Sie geben einen mathematischen Term ein, drücken Enter und erhalten das Ergebnis in der nächsten Zeile.

1.4.1 Operatoren und Terme

Ein mathematischer Term (Ausdruck) kann aus Zahlen, Operatoren und Klammern aufgebaut werden. Die Schreibweise ist manchmal ein kleines bisschen anders als in der Schulmathematik. Zum Beispiel dürfen Sie beim Multiplizieren den Multiplikationsoperator * niemals weglassen. Das Kommando

```
>>> (1 + 1) (6 – 2)
```

führt zu einer Fehlermeldung. Korrekt ist:

```
>>> (1 + 1) * (6 – 2)
```

```
8
```

Es gibt keine langen Bruchstriche. Für Zähler oder Nenner müssen Sie Klammern verwenden:

```
>>> (2 + 3) / 2
2.5
```

Python unterscheidet zwischen der exakten Division `/` und der ganzzahligen Division `//`. Die ganzzahlige Division liefert eine ganze Zahl, und zwar den nach unten gerundeten Quotienten. Probieren Sie aus:

```
>>> 3/2
1.5
>>> 3//2
1
```

Die Modulo-Operation% liefert den Rest, der bei einer ganzzahligen Division übrig bleibt. Beispiel: `7` geteilt durch `3` ist `2` Rest `1`.

```
>>> 7 // 3
2
>>> 7 % 3
1
```

Zum Potenzieren einer Zahl verwenden Sie den Operator `**`. Beachten Sie, dass Sie mit Python fast beliebig große Zahlen berechnen können.

```
>>> 2**3
8
>>> 2**-3
0.125
>>> 2**0.5
1.4142135623730951
>>> 137 ** 57
62097244310190258855130481009768710553783221891824568918264378730801621731509707020422858215922309341135893663853254591817
```

Bei Termen mit mehreren Operatoren müssen Sie deren Prioritäten beachten (Tabelle 1.2). Ein Operator mit höherer Priorität wird zuerst ausgewertet. Der Potenzoperator hat die höchste Priorität, Addition und Subtraktion die niedrigste.

```
>>> 2*3**2
18
>>> (2*3)**2
```

Operator	Bedeutung
`**`	Potenz, `x**y` = x^y
`*, /, //`	Multiplikation, Division und ganzzahlige Division
`%`	Modulo-Operation. Der Rest einer ganzzahligen Division.
`+, -`	Addition und Subtraktion

Tabelle 1.2: Arithmetische Operatoren in der Reihenfolge ihrer Priorität

1.4.2 Zahlen

Wer rechnet, verwendet Zahlen. Mit Python können Sie drei Typen von Zahlen verarbeiten:

- Ganze Zahlen (`int`)
- Gleitpunktzahlen (`float`)
- Komplexe Zahlen (`complex`)

Was ist überhaupt eine Zahl? In der Informatik unterscheidet man zwischen dem abstrakten mathematischen Objekt und der Ziffernfolge, die eine Zahl darstellt. Letzteres nennt man auch *Literal*. Für ein und dieselbe Zahl im mathematischen Sinne, sagen wir die `13`, gibt es unterschiedliche Literale, z.B. `13` oder `13.0` oder `13.0000`. Drei Schreibweisen – eine Zahl.

Ganze Zahlen (Typ int)

Literale für ganze Zahlen sind z.B. `12`, `0`, `-3`. Ganze Dezimalzahlen bestehen aus den zehn Ziffern `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`. Es kann ein negatives Vorzeichen vor die erste Ziffer

geschrieben werden. Eine ganze Dezimalzahl darf nicht mit einer Null beginnen. Probieren Sie es aus!

```
>>> 023
SyntaxError: invalid token
```

Ganze Zahlen dürfen bei Python beliebig lang sein. Eine Grenze ist nur durch die Speicherkapazität des Rechners gegeben. Probieren Sie es aus:

Ganze Zahlen sind vom Typ `int` (*engl.* integer = unversehrt, ganz). Mit der Funktion `type()` können Sie den Typ eines Literals abfragen:

```
>>> type(13)
<class 'int'>
>>> type(13.0)
<class 'float'>
```

Sie sehen, dass die Literale `13` und `13.0` zu verschiedenen Typen gehören, obwohl sie den gleichen mathematischen Wert darstellen. Was hat es mit dem Begriff `class` auf sich? Der Typ `int` wird durch eine Klasse (engl. *class*) realisiert. Eine Klasse kann man sich als eine Art Bauplan für Objekte eines Typs vorstellen. In der Klasse `int` ist zum Beispiel definiert, wie die Operationen für ganze Zahlen ausgeführt werden. Mehr Informationen zu Klassen finden Sie in Abschnitt 3.8.2.

Binär, oktal und hexadezimal – andere Schreibweisen für ganze Zahlen

Üblicherweise verwenden wir das dezimale Zahlensystem. Es gibt aber auch Binärzahlen (mit nur zwei Ziffern 0 und 1), Oktalzahlen (mit acht verschiedenen Ziffern) und Hexadezimalzahlen (mit 16 Ziffern).

Wie das Dezimalsystem ist auch das *Binärsystem* (Dualsystem, Zweiersystem) ein Stellenwertsystem für Zahlen. Aber anstelle von zehn Ziffern gibt es nur zwei, die Null und die Eins. Jede Zahl lässt sich als Summe von Zweierpotenzen (`1`, `2`, `4`, `8`, `16`, ...) darstellen. Die Binärzahl `10011` hat den Wert

```
1*16 + 0*8 + 0*4 + 1*2 + 1*1 = 16 + 2 + 1 = 19
```

Nun muss man natürlich erkennen können, ob eine Ziffernfolge wie `10011` als Dezimalzahl (zehntausendundelf) oder als Binärzahl gemeint ist. Deshalb beginnen bei Python Literale für Binärzahlen immer mit der Ziffer `0` und dem Buchstaben `b`, also z.B. `0b10011`. Wenn Sie in der Python-Shell eine solche Ziffernfolge eingeben und `Enter` drücken, erscheint in der nächsten Zeile der Wert als Dezimalzahl: