

## Kapitel 2

# Teleportation mit Variablen

Wir kommen nun zu dem, was wir uns eigentlich vorgenommen haben: Minecraft-Welten mit Python zu kontrollieren. Bist du bereit? In diesem Kapitel werden wir einen kleinen Rundgang durch die Grundlagen von Python machen. Danach wirst du deine neuen Fähigkeiten auf die Probe stellen und deine eigene Teleportationstour durch deine Minecraft-Welt erstellen.

Die in diesem Kapitel beschriebenen Konzepte sind nicht spezifisch für Minecraft-Python, können also in jedem Python-Programm verwendet werden, das du schreibst.

## Was ist ein Programm?

Ein *Programm* ist eine Abfolge von Anweisungen, die dafür sorgen, dass dein Computer eine oder mehrere bestimmte Aufgaben erledigt. Stell dir mal eine Stoppuhr-App auf deinem Smartphone vor. Eine solche App ist nichts anderes als ein Programm mit Anweisungen, die dem Smartphone sagen, was zu tun ist, wenn du auf den START- oder STOP-Button tippst. Sie umfasst ferner Anweisungen zum Anzeigen der verstrichenen Zeit auf dem Bildschirm. Irgendjemand hat diese Stoppuhr-App so programmiert, dass sie funktioniert.

In der ganzen Welt werden Tag für Tag Millionen von Programmen eingesetzt. Die Messaging-App auf deinem Handy ist ein Programm, Verkehrssampeln werden mit Programmen gesteuert, und sogar Computerspiele wie Minecraft sind Programme.

In diesem Buch lernst du die Grundlagen der Programmierung kennen und erfährst, wie man eigene Ideen in Minecraft durch das Schreiben von Programmen umsetzt.

## Daten in Variablen speichern

Beginnen wir mit der Speicherung von Daten in Variablen. *Variablen* gestatten dir das Speichern von Daten, die du an späterer Stelle im Programm noch einmal brauchst. *Daten* können dabei alles Mögliche sein, was gespeichert werden soll: Zahlenwerte, Namen, beliebige Texte, Objektlisten usw. Hier beispielsweise ist eine Variable namens `pickaxes`, die den Zahlenwert 12 speichert:

---

```
>>> pickaxes = 12
```

---

Variablen können Zahlen, Wörter und sogar ganze Sätze speichern, etwa »Hinfort mit dir, Creeper!«. Du kannst Variablenwerte auch ändern und damit in Minecraft einige nette Sachen machen. Und die Macht der Teleportation, wie wir sie gleich ausprobieren werden, basiert ebenfalls auf Variablen!

Zum Erstellen einer Variablen in Python brauchst du einen Variablennamen, ein Gleichheitszeichen (=) und einen Wert. Nehmen wir an, du bist im Begriff, dich auf eine lange Reise durch die unterschiedlichsten Minecraft-Biome zu machen, und möchtest dafür eine Menge Nahrungsmittel mitnehmen. Die Nahrungsmittel lassen sich als Variable darstellen. Beispielsweise ist in der folgenden Python-Shell `bread` der Variablenname und 145 der Wert:

---

```
>>> bread = 145
```

---

Der Variablenname steht immer links vom Gleichheitszeichen und der Wert, den du in der Variablen speicherst, steht rechts vom Gleichheitszeichen (siehe Bild 2.1). Diese Python-Codezeile *deklariert* die Variable `bread` (d.h. macht sie dem Computer bekannt) und nimmt eine *Zuweisung* des Werts 145 vor.

```
bread = 145
└──┬──┘ └──┬──┘
  Variablenname  Wert
```

*Bild 2.1:*  
Teile einer Variablendeklaration. Für 145 Brotlaibe musst du schon echt hungrig sein.

Nach Deklaration und Wertzuweisung kannst du den Namen der Variablen in der Python-Shell eingeben, um ihren Wert abzufragen:

---

```
>>> bread
145
```

---

Du kannst praktisch jeden Namen für eine Variable verwenden, doch bietet es sich an, eine aussagekräftige Bezeichnung zu vergeben, damit du nachvollziehen kannst, was in deinem Programm vor sich geht. Ebenfalls keine festgelegte Regel, aber trotzdem zu empfehlen ist eine Benennung mit einem Klein- statt einem Großbuchstaben am Anfang. An diese Vorgabe halten sich die meisten Python-Programmierer, und das solltest du auch tun, damit andere deinen Code leichter lesen können, falls du entscheiden solltest, ihn weiterzugeben.

#### Hinweis

Bitte beachte, dass der Wert einer Variablen *flüchtig* ist und nicht *gespeichert* wird. Der Variablenwert wird im Arbeitsspeicher des Computers abgelegt – wird das Programm beendet oder der Computer abgeschaltet, dann geht der Wert verloren. Schließe doch einmal IDLE, und starte es dann erneut. Was wird nun passieren, wenn du den Wert von *bread* abrufst?

## Programmiersprachen und ihre Struktur

Als *Syntax* werden die Regeln bezeichnet, die die Grammatik und Zeichensetzung einer Programmiersprache beschreiben – ähnlich der Grammatik und Zeichensetzung in der menschlichen Sprache. Wenn du die Syntax von Python verstanden hast, wirst du in der Lage sein, Programme zu schreiben, die ein Computer abarbeiten kann. Verwendest du jedoch eine fehlerhafte Syntax, dann wird der Computer nicht verstehen, was du ihm sagen willst.

Stell dir eine einzelne Anweisung in deinem Code als Satz vor. Im Deutschen steht am Ende eines Satzes ein Punkt. Python verwendet anstelle des Punkts einen Zeilenwechsel, um das Ende einer Anweisung und den Anfang der nächsten zu signalisieren. Jede Anweisung in einer neuen Zeile wird auch als *Statement* bezeichnet.

Nehmen wir nun beispielshalber an, du möchtest festlegen, wie viele Spitzhacken, Eisenerzblöcke und Bruchsteinblöcke du hast. In der Python-Shell könntest du nun Folgendes schreiben:

```
>>> pickaxes = 12
>>> iron = 30
>>> cobblestone = 25
```

Bild 2.2 zeigt, wie das in der Python-Shell aussieht.

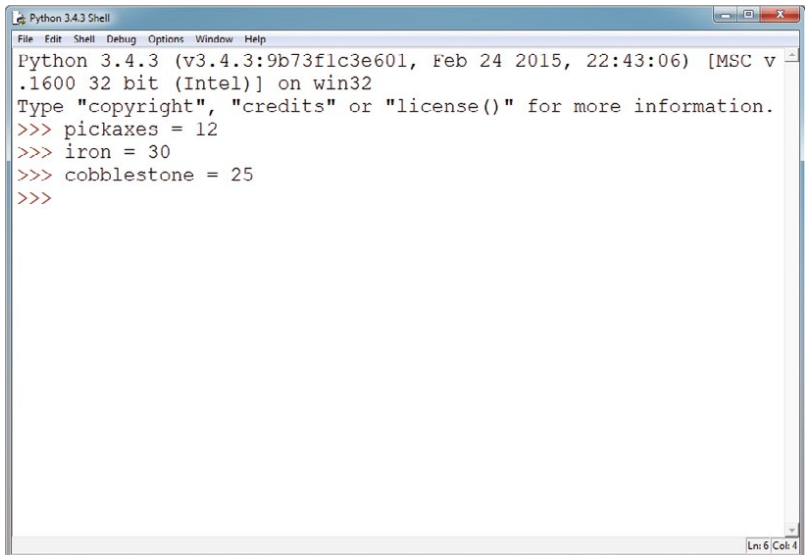


Bild 2.2: Codeeingabe in der Python-Shell

Beachte, dass jede Anweisung in einer eigenen Zeile steht. Durch die Zeilenwechsel weiß Python, dass du drei verschiedene Elemente mit Werten versiehst. Legst du hingegen nicht jede Anweisung in einer eigenen Zeile ab, dann verwirrt dies Python, und ein Syntaxfehler wird ausgegeben:

```
>>> pickaxes = 12 iron = 30 cobblestone = 25
SyntaxError: invalid syntax
```

Mit einem *Syntaxfehler* sagt Python dir, dass es etwas nicht verstanden hat. Python kann diese Anweisungen nicht verarbeiten, weil es nicht erkennen kann, wo eine Anweisung endet und wo die nächste beginnt.

Außerdem weiß Python auch nicht, was es machen soll, wenn du am Anfang einer Zeile ein Leerzeichen setzt:

---

```
>>> iron = 30
SyntaxError: unexpected indent
```

---

Wenn du genau hinsiehst, wirst du sehen, dass ich am Anfang der Zeile ein paar Leerzeichen gesetzt habe. Wenn du einen Syntaxfehler aufgrund eines unerwarteten Einzugs (engl. *unexpected indent*) erhältst, kannst du davon ausgehen, dass Leerzeichen am Anfang einer Zeile stehen, wo sie das nicht tun sollten.

Was das Schreiben von Code angeht, ist Python ausgesprochen pingelig. Wenn beim Eingeben der Beispiele aus diesem Buch ein Syntaxfehler auftritt, überprüfe deine Eingabe sorgfältig. Höchstwahrscheinlich hat sich irgendwo ein kleiner Fehler eingeschlichen.

## Syntaxregeln für Variablen

Einige wenige Syntaxregeln zur Benennung von Variablen musst du kennen, um Missverständnisse aufseiten von Python zu vermeiden.

- Python 3 erlaubt sogenannte Sonderzeichen in den Variablennamen (dazu gehören z. B. ä, ö oder ü), doch rate ich von deren Verwendung ab. Beschränke dich am besten auf die Buchstaben a–z, die Ziffern 0–9 und den Unterstrich (\_).
- Am Anfang eines Variablennamens darf keine Ziffer stehen. So etwas wie 9bread ist verboten. An allen anderen Stellen im Variablennamen dürfen Ziffern hingegen verwendet werden (z. B. bread9).
- Leerzeichen zu beiden Seiten des Gleichheitszeichens sind nicht erforderlich – dein Programm läuft auch ohne sie. Allerdings verbessern Leerzeichen die Lesbarkeit des Codes und sollten deswegen stets hinzugefügt werden.

Variablen sind etwas sehr Praktisches. Als Nächstes wirst du lernen, wie du den Wert einer Variablen ändern kannst. Dann bist du bereit für die Teleportation deines Spielers.

## Variablenwerte ändern

Du kannst den Wert einer Variablen jederzeit verändern. Dafür benutzt du die gleiche Anweisung wie beim Deklarieren. Angenommen, du triffst in Minecraft fünf Katzen und möchtest diesen Wert als Variable speichern.

Zunächst deklarierst du also eine Variable `cats` und weist ihr den Wert 5 zu. In der Python-Shell sähe das dann so aus:

---

```
>>> cats = 5
>>> cats
5
```

---

Später triffst du fünf weitere Katzen und möchtest den Wert dieser Variablen deswegen aktualisieren. Was passiert jetzt, wenn du den Wert von `cats` auf 10 setzt?

---

```
>>> cats = 10
>>> cats
10
```

---

Wenn du den neuen Wert von `cats` in Python abfragst, beträgt er nicht mehr 5! Würdest du die Variable `cats` in einem Programm verwenden, dann hätte diese nun den neuen Wert 10.

In Variablen lassen sich Daten aller Art ablegen. Am *Datentyp* erkennt der Computer, wie er mit einem bestimmten Datenelement umgehen muss. Ich werde zunächst einen Datentyp beschreiben, mit dem du es sehr oft zu tun bekommen wirst: Integer (ganze Zahlen). Im weiteren Verlauf des Kapitels kommen dann die Floats (Fließkommazahlen) hinzu.

## Integer

*Integer* sind positive oder negative ganze Zahlen. Werte wie 10, 32, -6, 194689 oder -5 sind Integer, Zahlen wie 3,14 und 6,025 jedoch nicht.

Wahrscheinlich verwendest du Integer-Zahlen Tag für Tag, ohne es überhaupt zu merken – sogar in Minecraft. Stell dir vor, du siehst zwölf Kühe auf einer Weide, während du unterwegs bist, um fünf Diamanten zu schürfen, und in deinem Inventar befinden sich zwei frische Äpfel. All diese Zahlen sind Integer.

Nehmen wir nun an, du besitzt in deiner Minecraft-Welt fünf Schweine und möchtest nun ein Programm schreiben, das die Anzahl der Schweine auf irgendeine Weise nutzt. In Python musst du eine Integer-Variable deklarieren, um die Anzahl der Schweine darzustellen (ich verwende hier englische Begriffe, du kannst natürlich deutsche verwenden):

---

```
>>> pigs = 5
```

---

Du kannst in Variablen auch negative Werte speichern. Um beispielsweise eine Temperatur von  $5^\circ$  unter null zu speichern, würdest du wie folgt deklarieren:

```
>>> temperature = -5
```

Um Python-Variablen und Integer mal mit Minecraft zu verwenden, musst du nun deine erste Mission absolvieren.

### Mission 1: Den Spieler teleportieren

Bei dieser Mission untersuchst du die Wirkungsweise von Variablen. Hierzu teleportierst du deinen Spieler mithilfe von Integern an eine andere Position.

Wie Bild 2.3 zeigt, hat dein Spieler in der Minecraft-Welt eine *Position*, die durch die drei *Koordinaten*  $x$ ,  $y$  und  $z$  dargestellt wird. Der Buchstabe  $y$  steht dabei für die Höhe;  $x$  und  $z$  stellen horizontale Positionen auf einer Ebene dar.

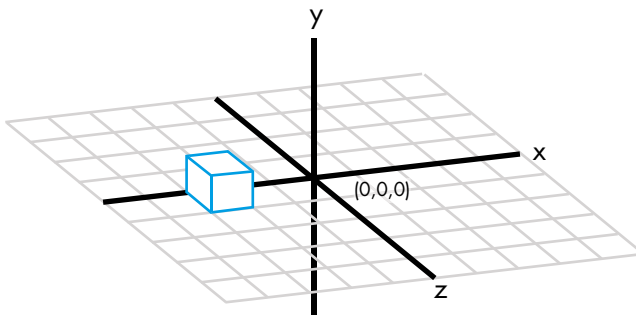


Bild 2.3: 3D-Koordinaten

Wenn du die Raspberry-Pi-Version des Spiels verwendest, wird die Position des Spielers durch drei Zahlen oben links in der Ecke des Spielfensters angegeben (Bild 2.4). In der Desktopversion kannst du die Koordinaten durch Drücken von **F3** anzeigen, indem du in der ersten Zeile des zweiten Textblocks links nach dem Eintrag *XYZ* suchst (Bild 2.5).

Bewege deinen Spieler im Spiel, und achte dabei darauf, wie sich die Positionszahlen verändern. Dies sollte beim Bewegen des Spielers in Echtzeit erfolgen. Eigentlich ganz cool, oder? Allerdings benötigt man hierbei für lange Wege viel Zeit. Wäre es nicht viel besser, wenn man – über Python –

direkt zu anderen Positionen springen könnte? Wir wollen uns einmal ansehen, wie das funktioniert.

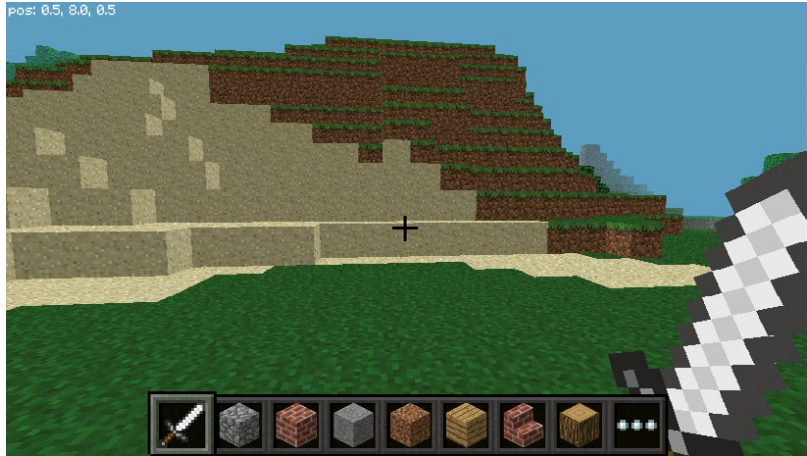


Bild 2.4: Spielerposition in der Minecraft: Pi Edition



Bild 2.5: Spielerposition in der Desktopversion von Minecraft

Schalte deinen Computer oder deinen Raspberry Pi ein, und führe die folgenden Schritte aus:

1. Öffne IDLE, und klicke auf **File ▶ New File** (auf einigen Computern musst du stattdessen **New Window** auswählen). Nun siehst du den



leeren Texteditor (Bild 2.6). Wenn du den Raspberry Pi verwendest oder wenn auf deinem Computer mehrere Python-Versionen installiert sind, dann verwende in jedem Fall Python 3, nicht Python 2.7.

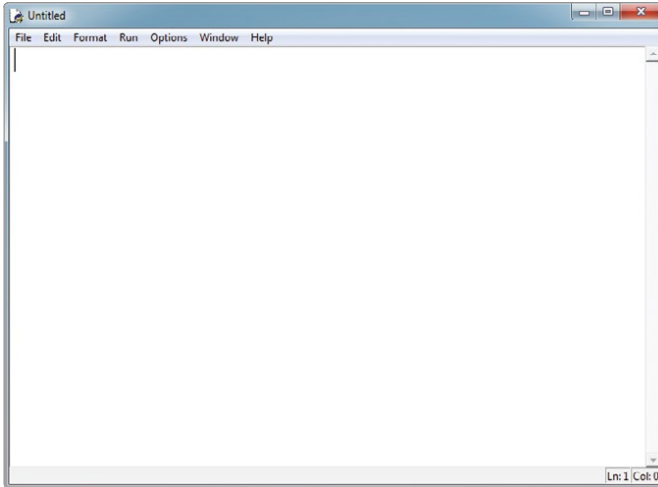


Bild 2.6: Neues Texteditorfenster in IDLE

2. Klicke nach dem Erscheinen des neuen Fensters auf **File ▶ Save As**.
3. Erstelle einen neuen Ordner namens *variables* in dem Ordner *Minecraft Python*, den du in Kapitel 1 angelegt hast.
4. Öffne den Ordner *variables*, gib deiner Datei den Namen *teleport.py*, und klicke auf **Save**.

Nun wechselst du in den IDLE-Texteditor und fügst dort die folgenden zwei Codezeilen am Anfang des Programms ein:

---

```
from mcpi.minecraft import Minecraft  
mc = Minecraft.create()
```

---

Über diese Zeilen verbindest du dein Programm mit Minecraft. Du musst sie in jedem Programm verwenden, das mit Minecraft interagieren soll. Als Nächstes erstellst du drei Integer-Variablen namens *x*, *y* und *z*:

---

```
x = 10  
y = 110  
z = 12
```

---

Diese Variablen stellen die Position dar, an die du deinen Spieler teleportieren möchtest. Lege sie zunächst wie hier gezeigt auf 10, 110 und 12 fest.

Danach gibst du die folgende Codezeile ein, die für den Transport des Spielers zuständig ist:

---

```
mc.player.setTilePos(x, y, z)
```

---

`setTilePos()` ist eine sogenannte *Funktion*, d. h. ein direkt einsatzbereites, wiederverwendbares Codeobjekt. Mit der Funktion `setTilePos(x, y, z)` wird Minecraft angewiesen, die Position des Spielers unter Verwendung der drei oben festgelegten Variablen zu ändern. Die Werte in den Klammern heißen *Argumente*. Du hast die von dir erstellten Variablen als Argumente an die Funktion *übergeben*, und deswegen kann sie die Werte von `x`, `y` und `z` bei der Ausführung verwenden.

### Achtung

Auf dem Raspberry Pi darfst du für die Variablen `x` und `z` keine Werte angeben, die größer als 127 oder kleiner als -127 sind. Die Minecraft-Welt auf dem Pi ist klein, und Werte außerhalb dieses Bereichs führen zu einem Absturz.

Listing 2.1 enthält den gesamten Code zum Teleportieren des Spielers, der auch in Bild 2.7 gezeigt ist:

*teleport.py*

---

```
❶ # Verbindung mit Minecraft herstellen
from mcpi.minecraft import Minecraft
mc = Minecraft.create()

# Variablen x, y und z zur Darstellung der Koordinaten festlegen
x = 10
y = 110
z = 12

# Position des Spielers ändern
mc.player.setTilePos(x, y, z)
```

---

Listing 2.1: Fertiger Teleportationscode


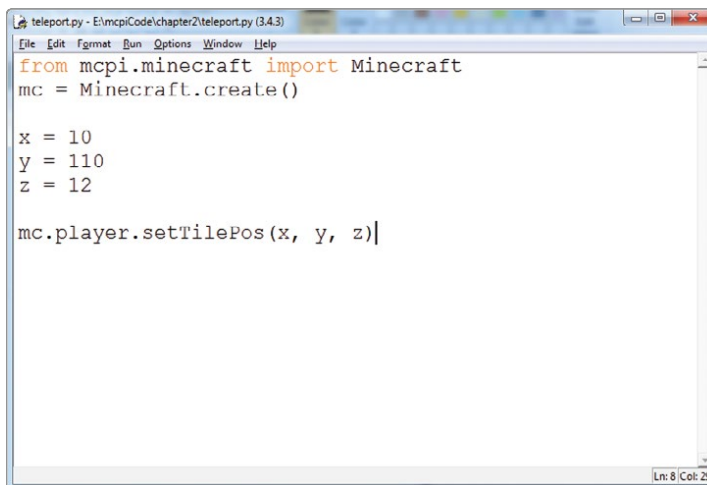
Damit du das Programm einfacher nachvollziehen kannst, habe ich einen *Kommentar*  ergänzt. Kommentare sind nützliche Anmerkungen, die beschreiben, was an der betreffenden Stelle im Code geschieht. Python ignoriert diese Anmerkungen jedoch. Mit anderen Worten: Wenn du das Programm ausführst, übergibt Python kommentierte Zeilen, ohne irgendetwas zu tun. Ein einzeliger Kommentar beginnt mit einer Raute (#). Meine Kommentare beschreiben, was die einzelnen Teile von *teleport.py* tun. Es ist eine gängige und empfohlene Praxis, den eigenen Code mit Kommentaren zu versehen. So kannst du dich später, wenn du deinen Code noch einmal liest, besser daran erinnern, was die einzelnen Teile des Programms tun sollen.

Bild 2.7 zeigt das vollständige Programm im IDLE-Texteditor.



```
teleport.py - E:\mcpicode\chapter2\teleport.py (3.4.3)
File Edit Format Run Options Window Help
from mcpi.minecraft import Minecraft
mc = Minecraft.create()

x = 10
y = 110
z = 12

mc.player.setTilePos(x, y, z)
```

Bild 2.7: Vollständiges Programm im IDLE-Texteditor

Nun wollen wir das Programm starten, um zu sehen, ob alles funktioniert. Führe dazu die folgenden Schritte aus:

1. Öffne Minecraft, indem du auf das Desktopsymbol klickst.
2. Wenn Du einen Raspberry Pi verwendest, klicke auf **Start Game** und dann auf **Create a New World**. In der Desktopversion von Minecraft öffnest du die Spielwelt wie in »Spigot ausführen und ein Spiel erstellen« auf Seite 9 für Windows und auf Seite 21 für den Mac beschrieben.
3. Nach dem Erstellen der Welt drückst du auf die **Esc**-Taste (bzw. beim Raspberry Pi auf die **Tabulatortaste**), um den Mauszeiger aus dem Spiel zu holen. Du kannst die Maus nun außerhalb des

Minecraft-Fensters bewegen. Wenn du das Spiel fortsetzen möchtest, doppelklickst du in das Minecraft-Fenster. Bild 2.8 zeigt das IDLE- und das Minecraft-Fenster auf meinem Computer.

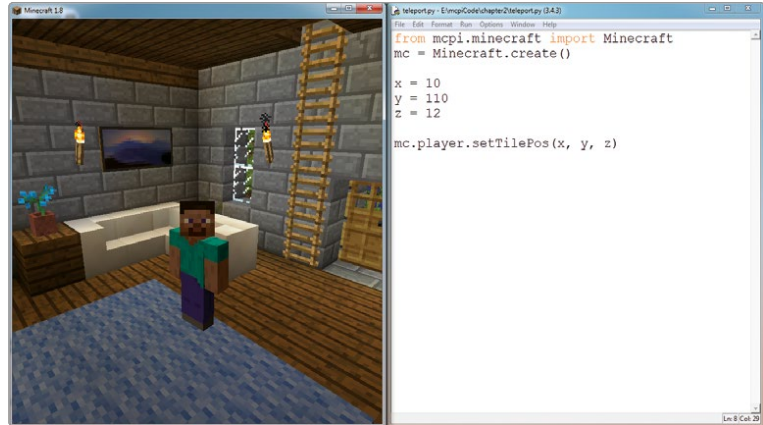


Bild 2.8: So sind das Minecraft-Fenster und der IDLE-Texteditor auf meinem Computer angeordnet.

4. Klicke auf das IDLE-Texteditorfenster mit deinem Programm *teleport.py*.
5. Klicke auf **Run ▶ Run Module** oder drücke **F5**. Falls du das Programm noch nicht gespeichert hast, wird IDLE dich immer fragen, ob du dies vor dem Ausführen tun möchtest. Klicke auf **OK**, um das Programm zu speichern. Wenn du auf **Cancel** klickst, wird das Programm nicht ausgeführt.

### Hinweis

Wenn du Programme aus IDLE auf einem Raspberry Pi startest, wird ein Dialogfeld angezeigt, das dich zum Speichern des Programms auffordert. Dieses Feld ist jedoch möglicherweise hinter dem Minecraft-Fenster verborgen. Wenn IDLE scheinbar abgestürzt ist, kann das schlicht und einfach daran liegen, dass das Dialogfeld nicht zu sehen ist. In diesem Fall musst du das Minecraft-Fenster minimieren und dann im IDLE-Dialogfeld auf **OK** klicken. Danach musst du das Minecraft-Fenster wieder maximieren.

Gut gemacht! Nun sollte dein Programm gestartet werden, und nach einigen Sekunden wird dein Spieler zu den Koordinaten (10, 110, 12) teleportiert (Bild 2.9). Deine Welt sieht nicht genauso aus wie meine, weswegen sich die Ausführung auf deinem Computer etwas anders gestalten wird.

### Bonus: Durch die Gegend springen

Glaubst du, du hast den Dreh mit der Teleportation raus? Ersetze  $x$ ,  $y$  und  $z$  durch andere Integer, und finde heraus, wo dein Spieler landet. Du kannst auch negative Werte ausprobieren!



Bild 2.9: Ich habe den Spieler von meinem Haus zur Position (10, 110, 12) teleportiert, die sich über einem Moor befindet. Schau mal nach unten!

## Float-Variablen

Nicht alle Zahlen sind ganze Zahlen. Zahlenwerte, die keine ganzen Zahlen (also Integer) sind, werden mit Nachkommastellen bzw. – im Englischen und damit auch in Python – mit Nachpunktstellen geschrieben. So kann man beispielsweise 0,5 Äpfel haben, was einem halben Apfel entspricht. Zahlen mit Nachkommastellen werden als *Fließkommazahlen* oder (auf Englisch) als *Floats* bezeichnet. Dies ist ein weiterer Datentyp, den Python verwendet. Fließkommazahlen werden anstelle von Integern benutzt, wenn eine größere Genauigkeit erzielt werden soll. Natürlich können Floats auch ganze Zahlen darstellen (z. B. 3.0), während Integer nicht in der Lage sind, Zahlen mit Nachkommastellen aufzunehmen.

Du hast vielleicht bemerkt, dass die Positionskordinaten deines Spielers in den Bildern 2.4 und 2.5 Dezimalstellen aufweisen – es sind also Floats!

In Python kannst du eine Float-Variable auf gleiche Weise deklarieren wie einen Integer-Wert. Um also etwa die Variable `x` auf 1.34 zu setzen, schreibst du:

---

```
>>> x = 1.34
```

---

Wichtig: Du musst einen »Punkt« (.) statt eines Kommas verwenden, sonst erhältst du einen Syntaxfehler – also »1.34« statt »1,34«. Das liegt daran dass im englischen Sprachraum der Punkt als Dezimaltrenner verwendet wird.

Willst du eine negative Float-Zahl erstellen, dann setzt du einfach das Minuszeichen (-) vor den Wert:

---

```
>>> x = -1.34
```

---

In der nächsten Mission erhältst du noch mehr Kontrolle über deine Teleportationskräfte, denn mithilfe von Floats werden wir unseren Spieler noch präziser in unserer Welt positionieren.

### Mission 2: Genau dort landen, wo man hinwill

Du weißt schon, wie du die Spielerposition mit Integer-Werten festlegen kannst. Mit Floats ist das aber noch genauer möglich. Bei dieser Mission werden wir das Programm aus Mission 1 so überarbeiten, dass der Spieler mithilfe eines Float-Werts teleportiert wird:

1. Öffne in IDLE das Programm *teleport.py* (Seite 44), indem du auf **File ▶ Open** klickst und die Datei in deinem Ordner *variables* auswählst.
2. Speichere eine Kopie dieses Programms als *teleportPrecise.py* in deinem Ordner *variables*.
3. Lege in der Datei *teleportPrecise.py* die Variablen *x*, *y* und *z* so fest, dass Floats anstelle von Integern verwendet werden. Setze also die Werte von *x*, *y* und *z* von 10, 110 und 12 auf 10.0, 110.0 und 12.0.
4. Ändere die letzte Codezeile in `mc.player.setPos(x, y, z)`. Hierzu musst du lediglich das Wort `Tile` entfernen.
5. Speichere das Programm.
6. Öffne eine Minecraft-Welt, und führe den Code aus.

Das endgültige Ergebnis sollte wie folgt aussehen:

---

```
# Verbindung mit Minecraft herstellen
from mcpi.minecraft import Minecraft
mc = Minecraft.create()
```

```
# Variablen x, y und z zur Darstellung der Koordinaten festlegen
x = 10,0
y = 110,0
z = 12,0
```

```
# Position des Spielers ändern
mc.player.setPos(x, y, z)
```

---

*teleportPrecise*  
*.py*

Beachte den Unterschied zwischen dem hier verwendeten `mc.player.setPos(x, y, z)` und der Variante `mc.player.setTilePos(x, y, z)` aus Listing 2.1. Die Funktion `setTilePos()` verwendet Integer, um die Position des Blocks anzugeben, zu dem du dich teleportieren lassen möchtest. `setPos()` funktioniert ein bisschen anders: Mithilfe von Floats wird nicht nur die Position des Blocks angegeben, zu dem du dich teleportieren lassen möchtest, sondern auch die exakte Position *auf* diesem Block. Mit meinem Programm habe ich mich auf die Spitze meines Turms teleportiert (Bild 2.10).



Bild 2.10: Mit Floats bin ich präzise auf der Spitze meines Turms gelandet.

### Bonus: Exaktes Teleportieren

Ändere die Werte der Variablen  $x$ ,  $y$  und  $z$ . Verwende dabei einen Mix aus positiven und negativen Werten. Starte das Programm. Nun ändere die Dezimalwerte geringfügig ab. Was geschieht?

## Teleportation mit dem time-Modul verlangsamen

Python führt deinen Code so schnell wie möglich aus. Du kannst Vorgänge allerdings auch verlangsamen, indem du deine Programme vor der Fortsetzung einige Sekunden warten lässt.

Um die Zeit in deinen Programmen zu beeinflussen, benötigst du das `time`-Modul, das eine Anzahl fertiger Funktionen für alles Mögliche umfasst, das mit dem Thema »Zeit« zu tun hat. Damit du das `time`-Modul verwenden kannst, musst du die folgende Codezeile ganz oben in deinen Programmen hinzufügen:

```
import time
```



Wenn du das `time`-Modul und die Funktion `sleep()` verwendest, die Bestandteil dieses Moduls ist, musst du in jedem Fall die richtige Reihenfolge beachten. Mit der `sleep()`-Funktion kannst du im Programm eine Pause mit einer bestimmten, in Sekunden angegebenen Länge festlegen. Du musst das `time`-Modul stets vor Verwendung der `sleep()`-Funktion importieren, sonst kann Python diese Funktion nicht finden und wird dadurch so sehr aus dem Tritt gebracht, dass es dein Programm nicht mehr weiter ausführt. Deswegen ist es immer am besten, alle im Code verwendeten Module stets zu Anfang zu importieren. Ich empfehle dir, alle `import`-Anweisungen am Beginn deines Programms zu gruppieren. Meine Struktur sieht beispielsweise so aus, dass ich zunächst die Codezeilen zum Herstellen der Verbindung mit Minecraft ganz oben einfüge und dass die `import time`-Anweisung dann in der dritten Zeile landet.

Hier siehst du ein Beispiel dafür, wie ich die `sleep()`-Funktion verwende:

---

```
time.sleep(5)
```

---

Mit dieser Codezeile unterbrichst du dein Programm für fünf Sekunden. Du kannst einen beliebigen Integer- oder Float-Wert angeben, wie auch das folgende Beispiel zeigt:

---

```
time.sleep(0.2)
```

---

Wenn ein Programm auf diese Codezeile stößt, wird es eine Pause von 0,2 Sekunden einlegen. Nachdem du nun gelernt hast, wie du zeitliche Abläufe steuern kannst, kommen wir zur nächsten Mission.

### Mission 3: Teleportationstour

Das Schöne an der Teleportation in Minecraft ist, dass du deinen Spieler hinschicken kannst, wohin du möchtest. Mit den Fähigkeiten, die du schon erlernt hast, wirst du deinen Spieler nun auf eine vollautomatische Reise durch deine gesamte Minecraft-Welt senden.

Bei dieser Mission wirst du das Ändern von Variablenwerten üben. Hierzu modifizieren wir den Code aus Mission 1 (Seite 41) so, dass der Spieler an mehrere Positionen teleportiert wird, die auf der gesamten Karte verteilt sind. Der Spieler wird also zunächst an eine Position geschickt, dann wartet das Programm einige Sekunden und teleportiert ihn weiter.

1. Öffne in IDLE das Programm *teleport.py* (Seite 44), indem du auf **File ▶ Open** klickst und die Datei in deinem Ordner *variables* auswählst.
2. Speichere eine Kopie dieses Programms als *tour.py* in deinem Ordner *variables*.
3. Füge direkt hinter dem Code, der dein Programm mit Minecraft verbindet, `import time` ein.
4. Am Ende des Programms fügst du `time.sleep(10)` hinzu.
5. Nun kopierst du die Zeilen mit den Variablen *x*, *y* und *z* und der Funktion `setTilePos()` in die Zwischenablage und fügst sie am Ende ein, sodass sie zweimal im Programm vorhanden sind.
6. Ändere die Werte der beiden Variablensätze *x*, *y* und *z* beliebig ab. Du kannst die Koordinaten jeder Position in deinem Spiel ermitteln, indem du dorthin gehst und die Koordinaten des Spielers so notierst, wie ich weiter oben beschrieben habe.
7. Speichere das Programm.
8. Öffne eine Minecraft-Welt, und führe den Code aus.

Das endgültige Ergebnis einschließlich der neu eingefügten Koordinaten sollte wie folgt aussehen:

*tour.py*

---

```
# Verbindung mit Minecraft herstellen
from mcpi.minecraft import Minecraft
mc = Minecraft.create()
import time

# Variablen x, y und z zur Darstellung der Koordinaten festlegen
x = Wert eintragen
y = Wert eintragen
z = Wert eintragen

# Position des Spielers ändern
mc.player.setTilePos(x, y, z)

# 10 Sekunden warten
time.sleep(10)

# Variablen x, y und z zur Darstellung der Koordinaten festlegen
x = Wert eintragen
y = Wert eintragen
z = Wert eintragen

# Position des Spielers ändern
mc.player.setTilePos(x, y, z)
```

---

Der Spieler sollte nun zur ersten Position teleportiert werden und nach einer zehnssekündigen Pause zur zweiten Position gelangen (Bild 2.11).

**Bonus: Noch mehr Teleportation**

Kopiere den Code von `tour.py`, um den Spieler beliebig häufig neu zu platzieren. Ersetze die 10 in der Funktion `time.sleep(10)` durch einen anderen Wert. Du kannst auch jeder `sleep()`-Funktion einen anderen Wert zuweisen, sodass der Spieler jedes Mal unterschiedlich lange wartet.

Danach probiere mal, bei jeder Änderung nur einen der Variablenwerte `x`, `y` und `z` zu ändern. Du musst nämlich gar nicht jede Variable jedes Mal neu zuweisen! Benutze abschließend auch einmal Floats anstelle von Integern.



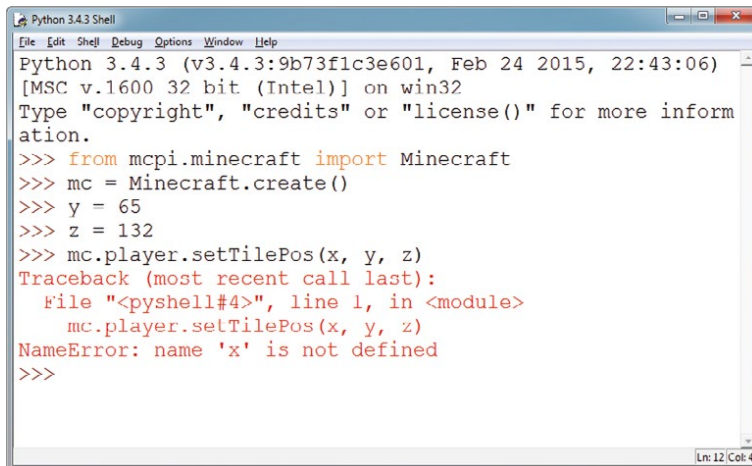
Bild 2.11: Ich habe die Koordinaten in meinem Programm so festgelegt, dass ich zunächst nach Hause und dann in die Wüste geschickt werde.

## Debuggen

Jeder macht mal einen Fehler, und auch die besten Programmierer bekommen ihren Code nicht unbedingt auf Anhieb zum Laufen. Ein lauffähiges Programm zu schreiben ist nur eine der Fähigkeiten, die einen guten Programmierer ausmachen. Ein guter Programmierer weiß auch, wie er Programmierfehler findet und loswird. Diesen Vorgang nennt man *Debuggen*, und einzelne Probleme in einem fehlerhaften Programm werden als *Bugs* (Wanzen) bezeichnet. In diesem Abschnitt wirst du einige

Tipps und Tricks lernen, damit du in der Lage bist, alle deine Programme künftig von Fehlern zu befreien.

Bugs können dafür sorgen, dass ein Programm entweder nicht mehr ausgeführt wird oder aber sich nicht so verhält, wie du es erwartest. Läuft ein Programm nicht, dann zeigt Python eine Fehlermeldung an. Bild 2.12 zeigt ein Beispiel.



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06)
[MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> from mcpi.minecraft import Minecraft
>>> mc = Minecraft.create()
>>> y = 65
>>> z = 132
>>> mc.player.setTilePos(x, y, z)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    mc.player.setTilePos(x, y, z)
NameError: name 'x' is not defined
>>>
```

Bild 2.12: Python zeigt eine Fehlermeldung an, weil ich mich nicht an die vorgeschriebene Syntax gehalten habe.

In Bild 2.12 ist zu sehen, dass ich etwas Code in die Python-Shell eingegeben habe, woraufhin mir eine Fehlermeldung angezeigt wird. Die Fehlermeldung enthält eine Menge Informationen, doch reicht es aus, die letzte Zeile (NameError: name 'x' is not defined) zu betrachten, um zu wissen, dass mit meiner Variablen x etwas nicht stimmt. Ich habe nämlich schlicht vergessen, die Variable x zu definieren. Zur Korrektur füge ich eine zusätzliche Codezeile ein, die die Variable x definiert:

---

```
>>> x = 10
```

---

Hierdurch wird dieser Fehler behoben, was aber noch nicht bedeutet, dass das ganze Programm fehlerfrei sein muss.

Bugs, die die Lauffähigkeit des Programms nicht beeinträchtigen, aber zu einem unerwünschten Verhalten führen, zeigen zwar keine Fehlermeldungen an, aber du wirst trotzdem schnell merken, dass irgendetwas nicht stimmt. Wenn du etwa eine Codezeile in deinen Teleportationsprogrammen vergisst (z. B. `setTilePos()`), dann wird das Programm zwar

fehlerfrei ausgeführt, aber die Position des Spielers ändert sich nicht. Und das ist für ein Teleportationsprogramm sicher kein erwünschtes Verhalten.

### Achtung

Zu den häufigsten Ursachen für Bugs gehören Tippfehler. Wenn du etwas so hinschreibst, dass der Computer nichts damit anfangen kann, führt das dazu, dass das Programm nicht läuft. Gehe sorgfältig vor, und vergewissere dich, dass die Schreibweise und die Groß-/Kleinschreibung korrekt sind!

### Mission 4: Bug im Teleportationsprogramm beheben

In dieser Mission werden wir zwei Programme berichtigen. Das erste Programm (Listing 2.2) ähnelt *teleport.py* (Seite 44), doch enthält diese Version fünf Fehler. Öffne eine neue Datei im IDLE-Texteditor, kopiere Listing 2.2 dorthin, und speichere das Ergebnis als *teleportBug1.py*.

*teleportBug1*  
*.py*

```
from mcpi.minecraft import Minecraft
# mc = Minecraft.create()

x = 10
y = 11
z = 12
```

Listing 2.2: *Schadhafte Version des Teleportationsprogramms*

Zum Debuggen dieses Programms führst du die folgenden Schritte aus:

1. Starte *teleportBug1.py*.
2. Sobald eine Fehlermeldung angezeigt wird, lies die letzte Zeile, um Informationen zum Fehler zu erhalten.
3. Behebe den Fehler, und führe den Code erneut aus.
4. Wiederhole die Fehlerkorrekturschritte so lange, bis das Programm den Spieler wie gewünscht an eine neue Position teleportiert.

**Hinweis**

Vergiss nicht, darauf zu achten, dass im Programm tatsächlich die Funktion `setTilePos()` aufgerufen wird!

Nun wollen wir ein anderes Programm debuggen. Die Version von *teleport.py* in Listing 2.3 wird zwar vollständig ausgeführt, doch erfolgt keine Teleportation des Spielers zur angegebenen Position. Kopiere Listing 2.3 in eine IDLE-Datei, und speichere sie als *teleportBug2.py*.

```
from mcpi.minecraft import Minecraft
mc = Minecraft.create()

x = 10
y = 110
z = -12

mc.player.setPos(x, z, y)
```

*teleportBug2*  
*.py*

Listing 2.3: *Teleportationsprogramm mit Bugs*

Anders als bei *teleportBug1.py* wird hier bei der Programmausführung keine Fehlermeldung angezeigt. Um dieses Programm zu korrigieren, musst du also den Code so lange unter die Lupe nehmen, bis du den Fehler gefunden hast. Das Programm soll den Spieler zur Position (10, 110, -12) bringen. Führe das Programm aus, und überprüfe die Koordinaten, an die der Spieler tatsächlich teleportiert wird. Diese Information kann beim Debuggen des Programms und bei der Erkennung des Problems hilfreich sein.

Wenn du alle Bugs in diesen beiden Programmen erwischst hast, solltest du jeden davon mit einem Kommentar versehen, um das jeweilige Problem zu erläutern. Wenn du dir die Probleme aufschreibst, die du beim Debuggen gefunden hast, wirst du die entsprechenden Fehler bei zukünftigen Projekten leichter erkennen oder sogar ganz vermeiden.

## Was du gelernt hast

Herzlichen Glückwunsch! Du hast nun deine ersten Python-Programme zur Steuerung eines Minecraft-Spielers mithilfe von Variablen und Funktionen geschrieben. Du hast außerdem zwei Datentypen (Integer und Floats) kennengelernt, und du weißt jetzt, wie man die Zeit steuern und Programme debuggen kann. Außerdem kennst du mit `setPos()` und `setTilePos()` bereits zwei nützliche Funktionen aus der Minecraft-Python-API.

In Kapitel 3 werden wir uns mit dem »Speed Building« befassen: also mit dem schnellen Platzieren von Blöcken mit mathematischen Operationen und Funktionen!

---