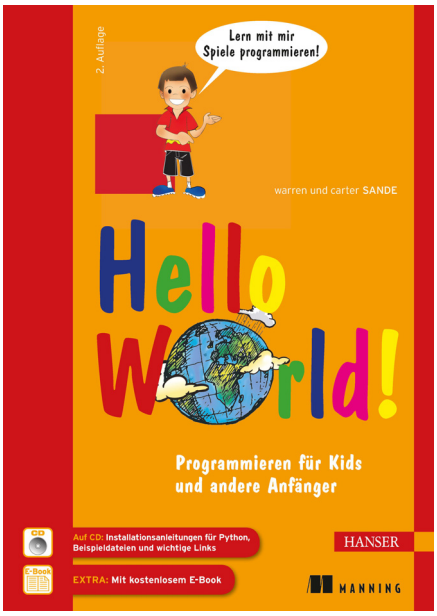
Hello World!

Programmieren für Kids und andere Anfänger

Bearbeitet von
Jürgen Dubau, Warren D. Sande, Carter Sande

2., aktualisierte und erweiterte Auflage 2014. Buch. 501 S.
ISBN 978 3 446 43806 4
Format (B x L): 18,7 x 24,4 cm
Gewicht: 1092 g

HANSER



Leseprobe

zu

„Hello World“ (2. Auflage)

von Warren und Carter Sande

ISBN (Buch): 978-3-446-43806-4
ISBN (E-Book): 978-3-446-43814-9

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/978-3-446-43806-4
sowie im Buchhandel

Kapitel 1

Erste Schritte

Wir werden die Programmiersprache Python benutzen, um Programmieren zu lernen. Um anfangen zu können, musst du also Python auf deinem Computer installieren. Erst danach kannst du lernen, wie es verwendet wird. Als Erstes werden wir Python einige Anweisungen geben und dann mehrere Anweisungen zusammenstellen, um ein Programm zu machen.

Python installieren

Das Erste, was du tun musst, ist Python auf dem Computer installieren, den du benutzen möchtest. Vielleicht ist Python ja bereits installiert, aber bei den meisten Leuten ist es das nicht. Also schauen wir uns jetzt an, wie es installiert wird.

Python ist ganz leicht zu installieren. Du suchst auf der beiliegenden CD die Version des Installationsprogramms, die zum Betriebssystem deines Computers passt.

Die gute alte Zeit

In der Frühzeit des Computerzeitalters hatten es die Menschen leicht: In die ersten PCs war oft eine Programmiersprache namens BASIC bereits eingebaut. Man musste nichts mehr installieren, sondern nur den Computer einschalten. Auf dem Bildschirm stand dann „READY“ und man konnte direkt anfangen, BASIC-Programme einzugeben. Klingt toll, nicht wahr?

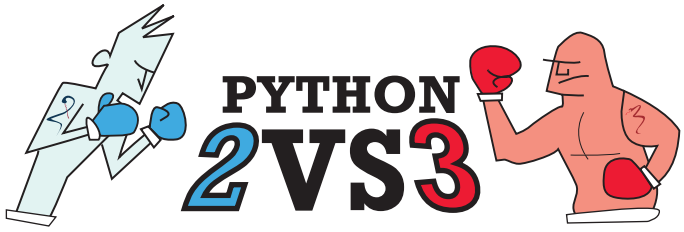
Aber dieses „READY“ war schon alles, was man bekam. Keine Programme, keine Fenster, keine Menüs. Für *alles*, was der Computer tun sollte, musste man zuerst ein Programm schreiben! Es gab noch keine Textverarbeitung, keinen Media Player, keine Browser, nichts von dem, an das wir heute gewöhnt sind. Es gab keine schicke Grafik und keinen Ton, nur gelegentlich einen Piepser, wenn man einen Fehler machte.

Es gibt Versionen für Windows, Mac OS X und Linux. Alle Beispiele in diesem Buch verwenden Windows, aber mit Mac OS X oder Linux funktioniert Python fast genauso. Befolge einfach die Anleitung auf der Website, um den richtigen Installer für dein System auszuführen.

Wenn Python auf deinem Computer bereits installiert ist und du den Installer nicht brauchst, musst du trotzdem darauf achten, dass auch die „Extras“, die für dieses Buch nötig sind, installiert werden. Wie das geht, steht bei den Installationsanweisungen auf der CD.

Wenn du Windows verwendest, klickst du auf „Windows Installer“. Dein Computer beginnt dann, die Python-Software herunterzuladen. Im Fenster „License Agreement“ klickst du auf das obere Optionsfeld („I accept …“) und dann einfach immer weiter auf „Next“ und schließlich auf „Install“. Wenn du Probleme damit hast, findest du bestimmt einen Erwachsenen, der dir gerne dabei hilft, vielleicht deine Eltern oder Lehrer.

In diesem Buch werden wir die Python-Version 2.5 verwenden. Wenn du den Installer von der beiliegenden CD benutzt, bekommst du genau dieselbe Version. Wenn du dies hier liest, gibt es aber vielleicht schon neuere Versionen von Python. Alle Beispiele in diesem Buch wurden mit Python 2.5 getestet. Wahrscheinlich werden sie auch mit späteren Versionen noch funktionieren, aber da ich nicht in die Zukunft sehen kann, kann ich das auch nicht garantieren.



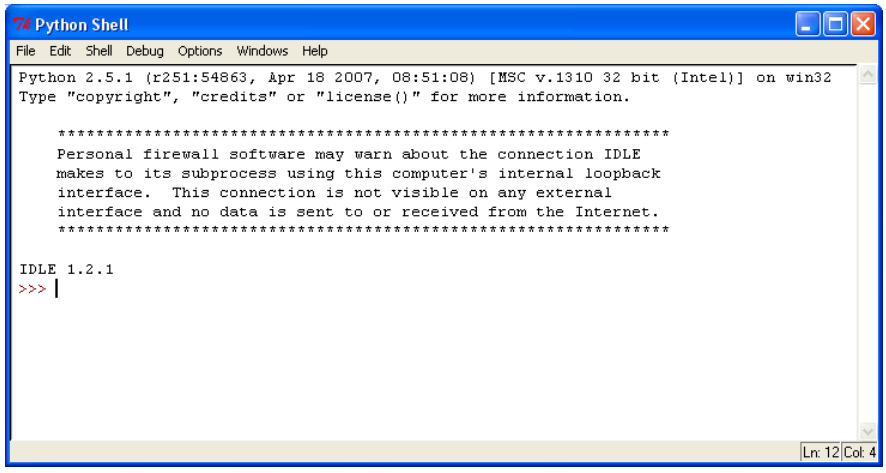
Python 2 versus Python 3

Ein paar Jahre, bevor dieses Buch geschrieben wurde, erschien die neue Version von Python: Python 3. Es stellte sich allerdings heraus, dass es gar keine „Aktualisierung“ war, sondern eher eine Art Weggabelung. Das heißt, viele wollten nicht zu Python 3 wechseln, sondern machten mit Python 2 weiter. Die Leute, die Python entwickeln, haben sowohl neue Versionen von Python 2 als auch von Python 3 produziert. Zu der Zeit, als diese Neuauflage von „Hello World!“ geschrieben wurde, waren die beiden aktuellen Versionen Python 2.7.3 und Python 3.3.0. In diesem Buch wird Python 2.7.3 verwendet, und der Code wird ziemlich wahrscheinlich mit zukünftigen Versionen von Python 2.x kompatibel sein. Weitere Einzelheiten über Python 2 im Vergleich zu Python 3 findest du in Anhang B.

Python mit IDLE starten

Python kann auf verschiedene Arten gestartet werden. Wir verwenden dazu zunächst ein Programm namens IDLE (sprich „Eidel“).

Im **Start**-Menü siehst du unter **Python 2.5** den Eintrag **IDLE (Python GUI).** Wenn du darauf klickst, öffnet sich das IDLE-Fenster. Es müsste ungefähr wie das hier gezeigte Fenster aussehen.



IDLE ist eine so genannte Python-*Shell*. Eine Shell gibt dir die Möglichkeit, mit einem Programm zu arbeiten, indem du Text eintippst, und diese Shell hier ist eben für Python da. (Darum steht auch „Python Shell“ in der Titelleiste des Fensters.) IDLE ist außerdem eine grafische Benutzeroberfläche (englisch „GUI“), und deshalb steht der Eintrag **Python GUI** im **Start**-Menü. Außer der Shell hat IDLE noch einiges andere zu bieten, aber dazu kommen wir später.

Worterklärung

GUI bedeutet *„Grafische Benutzeroberfläche“*. Das ist ein Bildschirm mit Fenstern, Menüs, Buttons, Bildlaufleisten usw. Programme, die keine GUI haben, nennt man *Konsolenprogramme* oder *Kommandozeilenprogramme*.

Das „**>>>**“ im obigen Bild ist die *Eingabeaufforderung* von Python. Eine Eingabeaufforderung ist das, was ein Programm anzeigt, wenn es darauf wartet, dass du etwas eintippst. Das „**>>>**“ zeigt dir, dass Python bereit ist, Anweisungen von dir zu bekommen.

Anweisungen bitte

Geben wir Python jetzt unsere erste Anweisung. Setze den Cursor hinter das „>>>“-Zeichen und gib Folgendes ein:

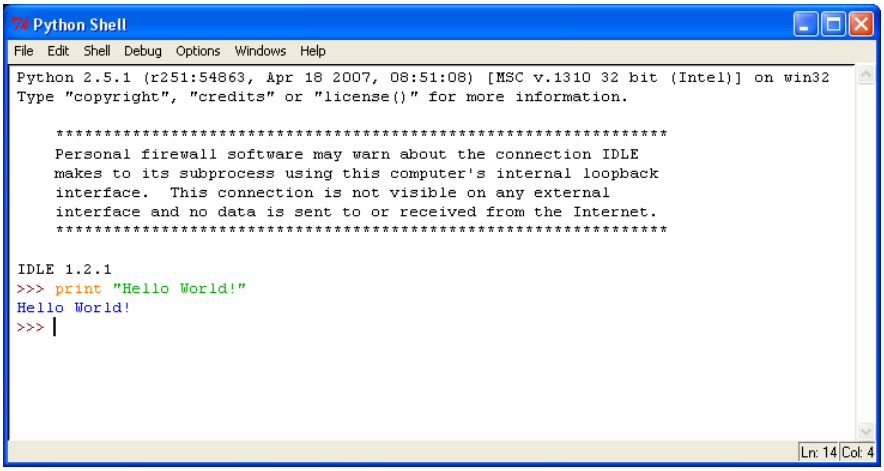
```
print "Hello World!"
```

Dann drücke auf die Eingabetaste (auf manchen Tastaturen heißt sie auch „Return“). Nach jeder eingegebenen Zeile musst du auf diese Taste drücken.

Nachdem du die Eingabetaste gedrückt hast, müsste folgende Antwort erscheinen:

```
Hello World!
>>>
```

Das folgende Bild zeigt, wie das im IDLE-Fenster aussieht.



Python tat, was du ihm gesagt hast: Es gab deine Meldung aus. (Beim Programmieren bedeutet **`print`** oft, dass ein Text auf dem Bildschirm ausgegeben und nicht auf Papier gedruckt wird.) Diese eine Zeile ist eine Python-*Anweisung*. Und jetzt bist du schon auf dem Weg zum Programmierer! Der Computer steht unter deinem Befehl!

Übrigens ist es für angehende Programmierer Tradition, den Computer als Erstes „Hello World!“ auf dem Bildschirm ausgeben zu lassen. Hier hat auch der Titel dieses Buches seinen Ursprung. Auch du folgst dieser Tradition. Willkommen in der Welt der Programmierung!

Gute Frage! IDLE möchte uns helfen, die Dinge besser zu verstehen. Es zeigt Sachen in verschiedenen Farben an, damit wir die verschiedenen Teile des Codes besser unterscheiden können. (*Code* ist nur ein anderes Wort für die Anweisungen, die du dem Computer in einer Sprache wie Python erteilst.) Ich werde im weiteren Verlauf des Buches erklären, was die Teile des Codes tun.

Wenn es nicht funktioniert

Wenn du etwas falsch gemacht hast, siehst du vielleicht so etwas wie dieses:

```
>>> pront "Hello World!"
SyntaxError: invalid syntax
>>>
```

Diese Fehlermeldung bedeutet, dass du etwas eingegeben hast, was Python nicht versteht. Im obigen Beispiel wurde **`print`** versehentlich als **`pront`** geschrieben. Damit kann Python nichts anfangen. Wenn dir so etwas passiert, versuche es noch einmal und achte darauf, alles genau wie im Beispiel zu schreiben.

Stimmt – weil **`print`** in Python ein Schlüsselwort ist und **`pront`** nicht.

Worterklärung

Ein *Schlüsselwort* ist ein besonderes Wort, das zur Sprache Python gehört. Man nennt es auch „*reserviertes* Wort“.

Mit Python interagieren

Soeben hast du Python im interaktiven Modus verwendet. Du hast einen Befehl (eine Anweisung) eingegeben und Python hat ihn *sofort* ausgeführt.

Worterklärung

Du kannst einen Befehl, oder eine Anweisung oder ein Programm auf dem Computer *ausführen*. Das ist nur ein anderes Wort für „laufen lassen".

Wir wollen noch etwas anderes im interaktiven Modus versuchen. Gib an der Eingabeaufforderung Folgendes ein:

```
>>> print 5 + 3
```

Herauskommen müsste Folgendes:

```
8
>>>
```

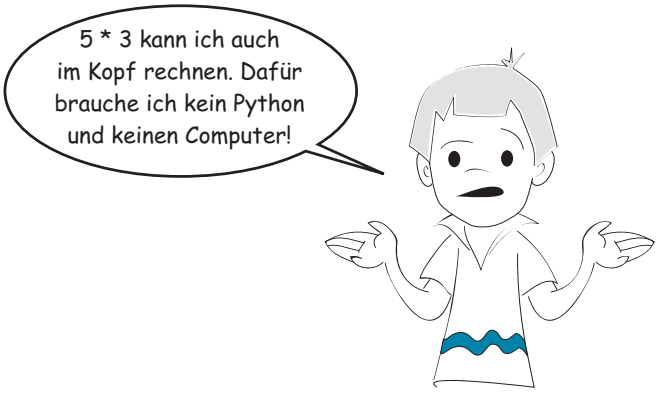
Also kann Python auch addieren! Das ist allerdings kein Wunder, denn Computer sind gut im Rechnen.

Probieren wir noch etwas aus:

```
>>> print 5 * 3
15
>>>
```

In fast allen Computerprogrammen und -sprachen wird das Symbol * zum Multiplizieren gebraucht. Dieses Zeichen nennt man *Sternchen*.

Wenn du in Mathe daran gewöhnt bist, „fünf mal 3" als „5 x 3" zu schreiben, musst du dich in Python auf das Symbol * umstellen. (Drücke einfach die Umschalttaste und +, links neben der Eingabetaste).



Gut, aber wie wär's hiermit:

```
>>> print 2345 * 6789
15920205
>>>
```

Gut, aber wie wär's hiermit:

```
>>> print 12345678987654321234567890 * 98765432123456789876543210
12193263200731596000609652202408166072245112635269
>>>
```

Stimmt. Mit dem Computer kannst du auch mit sehr, sehr großen Zahlen rechnen.

Du kannst auch dieses hier tun:

```
>>> print "Katze" + "Hund"
KatzeHund
>>>
```

Oder versuch's mal damit:

```
>>> print "Hallo " * 20
Hallo Hallo Hallo Hallo Hallo Hallo Hallo Hallo Hallo Hallo
Hallo Hallo Hallo Hallo Hallo Hallo Hallo Hallo Hallo Hallo
```

Außer Mathe können Computer noch eine andere Sache ziemlich gut: Sie können etwas wieder und wieder tun. Hier haben wir Python gesagt, es soll zwanzigmal „Hallo" ausgeben.

Später werden wir noch mehr im interaktiven Modus tun, aber jetzt ist es …

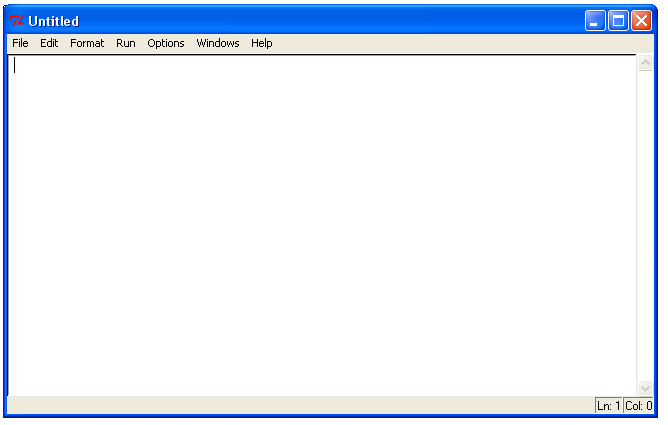
Programmierzeit

Unsere bisherigen Beispiele waren einfache Python-Anweisungen (im interaktiven Modus). So lässt sich zwar einiges herausfinden, was Python kann, aber diese Beispiele sind noch keine richtigen Programme. Wie schon gesagt besteht ein Programm aus mehreren Anweisungen, die zusammengefasst werden. Schreiben wir also unser erstes Python-Programm.

Wenn ich über Menübefehle rede, wie zum Beispiel **File > New**, dann ist der erste Teil (hier **File**) der Eintrag im Hauptmenü. Das **>** sagt dir, dass der nächste Teil (hier **New**) ein Untereintrag aus dem File-Menü ist. Diese Schreibweise werde ich im gesamten Buch verwenden.

Zuerst brauchst du eine Möglichkeit, um unser Programm einzugeben. Wenn du es einfach in das interaktive Fenster eintippst, wird Python es sich nicht „merken". Du brauchst also einen Texteditor (wie Notepad für Windows oder TextEdit für Mac OS X), der das Programm auf der Festplatte speichern kann. IDLE enthält bereits einen Editor, der für deine Zwecke viel besser ist als Notepad. Um ihn zu finden, wählst du aus dem IDLE-Menü **File > New Window**.

Du siehst dann ein Fenster wie in diesem Bild. Auf der Titelleiste steht „Untitled" („ohne Titel"), weil du ihm noch keinen Namen gegeben hast.



Gib jetzt das Programm aus Listing 1.1 in den Editor ein:

Listing 1.1 Unser erstes richtiges Programm

```
print "Ich liebe Pizza!"
print "Pizza " * 20
print "lecker " * 40
print "Ich platze gleich."
```

Hast du gemerkt, dass der Titel „Listing 1.1“ ist? Wenn der Beispielcode ein komplettes Programm ergibt, werde ich ihn auf diese Weise nummerieren, damit du ihn im **\Beispiele-Ordner** oder auf der Website leicht wiederfinden kannst.

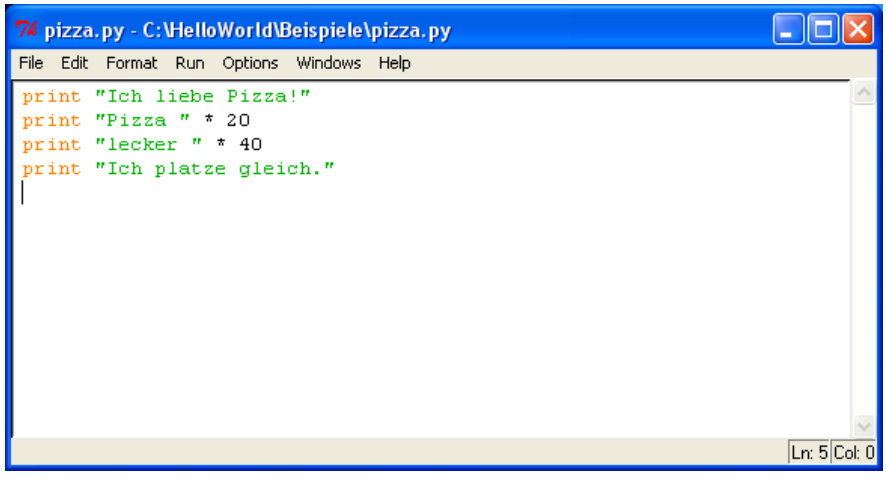
Wenn du fertig bist, speicherst du das Programm mit dem Menübefehl **File > Save** oder **File > Save as**. Nenne die Datei **pizza.py**. Du kannst sie speichern, wo du willst (du musst dir allerdings merken, wo sie ist, damit du sie später noch wiederfindest). Am besten erstellst du einen neuen Ordner für deine Python-Programme. Der „.py“-Teil am Ende ist wichtig, weil er dem Computer sagt, dass dies ein Python-Programm ist und nicht bloß irgendeine Textdatei.

Vielleicht hast du bemerkt, dass der Editor im Programm andere Farben verwendet. Manche Wörter sind orange und andere sind grün. Das liegt daran, dass der IDLE-Editor schon davon ausgeht, dass du ein Python-Programm eingibst. In Python-Programmen zeigt der Editor Python-Schlüsselwörter in orange, und alles, was in Anführungszeichen steht, in grün. Das soll dir helfen, Python-Code leichter zu lesen.

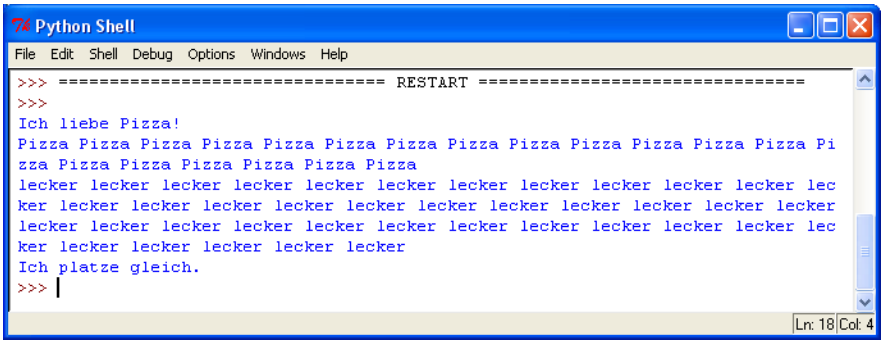
Manche Versionen von IDLE zeigen die Farben erst dann an, wenn das Programm als **.py**-Datei abgespeichert wurde, wie zum Beispiel **pizza.py**.

Das erste Programm ausführen

Sobald du das Programm gespeichert hast, gehst du in das **Run**-Menü (immer noch im IDLE-Editor) und klickst auf **Run Module** (wie im nächsten Bild). Dann läuft dein Programm.



Du wirst sehen, dass das Fenster der Python-Shell (das Fenster, das beim Starten von IDLE erschienen ist) wieder aktiv wird und so etwas wie dieses hier zeigt:



Das **RESTART** bedeutet, dass du ein Programm gestartet hast. (Dies wird nützlich sein, wenn du Programme wiederholt ausführst, um sie zu testen.)

Dann läuft das Programm ab. Naja, so richtig viel tut es noch nicht. Aber du hast den Computer dazu gebracht, zu tun, was du ihm gesagt hast. Unsere Programme werden im Laufe der Zeit immer interessanter werden.

Wenn etwas schiefgeht

Was passiert, wenn du einen Fehler im Programm machst und es nicht läuft? Es können zwei verschiedene Arten von Fehlern auftreten. Beide wollen wir uns anschauen, damit du weißt, welcher in deinem Fall passiert ist.

Syntaxfehler

IDLE prüft dein Programm, bevor es versucht, es auszuführen. Wenn IDLE einen Fehler findet, ist das normalerweise ein *Syntaxfehler*. Syntax ist die Rechtschreibung und Grammatik in einer Programmiersprache, also bedeutet ein Syntaxfehler, dass du etwas geschrieben hast, das kein richtiger Python-Code ist.

Hier folgt ein Beispiel.

Übrigens: Alles, was mit dem Zeichen # anfängt, ist ein *Kommentar*. Kommentare musst du nicht mit eintippen, weil sie nur für dich zum besseren Verständnis da sind. Für das Programm spielen sie keine Rolle. Wenn du mehr über Kommentare wissen möchtest, kannst du in Kapitel 9 nachschlagen.

```
print "Hallo, und willkommen bei Python!"
print "Ich hoffe, es wird dir Spaß machen, Programmieren zu lernen."
print Bis bald!"
      ↑ Anführungszeichen fehlt
```

Wir haben ein Anführungszeichen zwischen **`print`** und **`Bis bald!"`** vergessen.

Wenn du versuchst dieses Programm auszuführen, würde IDLE eine Nachricht einblenden, die besagt: „There's an error in your program: invalid syntax" (deutsch: „Fehler im Programm: Ungültige Syntax.") Dann würdest du in deinen Code schauen, um zu sehen, was verkehrt ist. IDLE würde (in rot) die Stelle markieren, an der es den Fehler gefunden hat. Das muss nicht immer genau die Stelle sein, die das Problem verursacht, dürfte aber zumindest in der Nähe liegen.

Laufzeitfehler

Die zweite Fehlersorte ist eine, die Python (oder IDLE) erst entdecken kann, wenn das Programm läuft. Deshalb wird dieser Fehlertyp *Laufzeitfehler* genannt. Hier ist ein Beispiel für einen Laufzeitfehler in einem Programm:

```
print "Hallo, und willkommen bei Python!"
print "Ich hoffe, es wird dir Spaß machen, Programmieren zu lernen."
print "Bis bald!" + 5
```

Wenn wir das speichern und versuchen, es auszuführen, beginnt das Programm auch tatsächlich zu laufen. Die ersten beiden Zeilen werden ausgegeben, aber dann folgt eine Fehlermeldung (achte auch hier wieder auf die Kommentare):

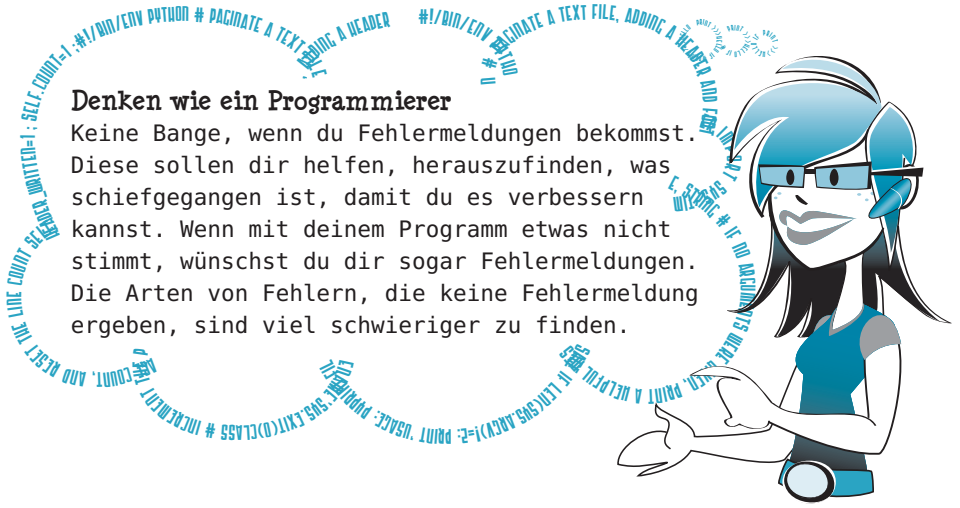
```
>>> ============================ RESTART ============================
>>>
Hallo, und willkommen bei Python!
Ich hoffe, es wird dir Spaß machen, Programmieren zu lernen.

Traceback (most recent call last):  <-- Beginn der Fehlermeldung
  File "C:/HelloWorld/Beispiele/error1.py", line 3, in <module>  <-- Ort des Fehlers
    print "Bis bald!" + 5  <-- Die „verkehrte" Codezeile
TypeError: cannot concatenate 'str' und 'int' objects  <-- Der Fehler laut Python
>>>
```

Die Zeile, die mit **`Traceback`** beginnt, ist der Anfang der Fehlermeldung. Die folgende Zeile sagt dir den Namen der Datei und die Nummer der Zeile, wo der Fehler passiert ist. Dann wird die fehlerhafte Codezeile angezeigt, damit du besser erkennen kannst, wo in deinem Code das Problem steckt. Der letzte Teil der Fehlermeldung sagt dir, was nach Pythons Meinung falsch ist. Wenn du erst mehr über Programmieren und Python weißt, wird es leichter für dich, Fehlermeldungen zu verstehen.

Nun, Carter, das ist wie in dem Sprichwort, dass man nicht Äpfel mit Birnen vergleichen kann. In Python kannst du keine ganz unterschiedlichen Sachen addieren, wie zum Beispiel Zahlen und Text. Darum ergab **`print "Bis bald!" + 5`** einen Fehler. Es ist, als würde ich sagen: „Wieviel ergeben 5 Äpfel plus 3 Birnen?".

Das ergibt 8, aber 8 wovon? Diese Dinge zu addieren ergibt eigentlich keinen Sinn. Aber es ist fast immer möglich, etwas mit einer Zahl zu multiplizieren, um mehr von der betreffenden Sache zu erhalten. (Wenn du 2 Äpfel mit 5 multiplizierst, hast du 10 Äpfel.) Das ist der Grund, weshalb `print "Bis bald!" * 5` funktioniert.



Denken wie ein Programmierer

Keine Bange, wenn du Fehlermeldungen bekommst. Diese sollen dir helfen, herauszufinden, was schiefgegangen ist, damit du es verbessern kannst. Wenn mit deinem Programm etwas nicht stimmt, wünschst du dir sogar Fehlermeldungen. Die Arten von Fehlern, die keine Fehlermeldung ergeben, sind viel schwieriger zu finden.

Tipp: Wenn man Python keine besonderen Anweisungen gibt, dann kann es mit den deutschen Umlauten (ä, Ä, ü, Ü, ö, Ö) und dem ß nicht richtig umgehen, sondern gibt eine Fehlermeldung („no encoding declared“) aus.

Das bedeutet, dass Python nicht weiß, wie der Text gespeichert worden ist. Um solche Zeichen trotzdem verwenden zu können, kannst du Python diese Codierung (englisch „Encoding“) mitteilen. Das machst du, indem du diese Zeile an erster Stelle im Code schreibst:

```
# -*- coding: iso-8859-1 -*-
```

Es gibt auch noch weitere Codierungen, doch sehr wahrscheinlich ist diese Zeile genau die richtige für dich.

Die Programme im Beispiel-Verzeichnis enthalten diese Zeile immer. Im Buch haben wir sie aber oft weggelassen, um etwas Platz zu sparen.

Denk bitte beim Abtippen daran!

Dein zweites Programm

Das erste Programm hat noch nicht viel getan. Es hat nur etwas auf dem Bildschirm ausgegeben. Wir wollen jetzt etwas Interessanteres versuchen.

Das nächste Programm in Listing 1.2 ist ein Spiel, bei dem du Zahlen raten musst. Öffne im IDLE-Editor mit **File > New Window** eine neue Datei, gib den Code aus Listing 1.2 ein (wieder ohne die Kommentare) und speichere ihn. Du kannst dem Programm einen beliebigen Namen geben, wenn er nur auf .py endet. Bei mir heißt es **ZahlenRaten.py**.

Das Programm hat nur 18 Zeilen Python-Anweisungen plus einige Leerzeilen zur Verbesserung der Lesbarkeit. Es dürfte relativ schnell einzugeben sein. Keine Sorge, weil wir noch nicht alle Bedeutungen dieses Codes besprochen haben. Das werden wir sehr bald nachholen.

Listing 1.2 Zahlen raten

```
import random
geheimnis = random.randint(1, 99)  <-- Wählt eine geheime Zahl aus
tipp = 0
versuche = 0
print "AHOI! Ich bin der berüchtigte Kapitän Hook und habe ein "Geheimnis!"
print "Es ist eine Zahl zwischen 1 und 99. Du hast 6 Versuche."
while tipp != geheimnis and versuche < 6:
    tipp = input("Was rätst du? ")  <-- Lässt den Spieler raten
    if tipp < geheimnis:
        print "Zu niedrig, du Landratte!"
    elif tipp > geheimnis:
        print "Zu hoch, du Leichtmatrose!"

    versuche = versuche + 1  <-- Ein Versuch ist verbraucht

if tipp == geheimnis:
     print "Ha! Du hast es! Hast mein Geheimnis erraten!"
else:
    print "Alle Versuche verbraucht! Mehr Glück beim nächsten Mal!"
    print "Die Geheimzahl war ", geheimnis
```

Bis zu 6 Versuche

Nachricht am Spielende

Wenn du das eingibst, achte bitte auf die Einrückung hinter der `while`-Anweisung und die zusätzliche Einrückung der Zeilen hinter `if` und `elif`. Beachte bitte auch die Doppelpunkte am Ende einiger Zeilen. Wenn du den Doppelpunkt an der richtigen Stelle eingibst, hilft dir der Editor, indem er die nächste Zeile für dich einrückt.

Speichere das Programm und führe es mit **Run > Run Module** aus, wie schon das erste Programm. Versuche, das Spiel zu spielen und schau, was passiert. Hier siehst du zum Beispiel eines meiner Spiele:


```
>>> ====================== RESTART ======================
>>>
AHOI! Ich bin der berüchtigte Kapitän Hook und habe ein Geheimnis!
Es ist eine Zahl zwischen 1 und 99. Du hast 6 Versuche.
Was rätst du? 40
Zu hoch, du Leichtmatrose!
Was rätst du? 20
Zu hoch, du Leichtmatrose!
Was rätst du? 10
Zu niedrig, du Landratte!
Was rätst du? 11
Zu niedrig, du Landratte!
Was rätst du? 12
Ha! Du hast es! Hast mein Geheimnis erraten!
>>>
```

Ich musste fünfmal raten, um die Geheimzahl zu finden. Es war die 12.

In den nächsten Kapiteln werden wir alles über die Anweisungen **`while`**, **`if`**, **`else`**, **`elif`** und **`input`** lernen. Aber wahrscheinlich hast du schon die Grundidee verstanden, nach der dieses Programm funktioniert.

- Das Programm wählt eine beliebige Geheimzahl aus.
- Der Benutzer gibt seine geratenen Zahlen ein.
- Das Programm prüft jedesmal, ob die Zahl höher oder niedriger als die Geheimzahl ist.
- Der Benutzer versucht es so lange, bis er die Zahl rät oder keine Versuche mehr übrig hat.
- Wenn er die richtige Zahl geraten hat, hat er gewonnen.

0011000111001110000110110100011011010111001100011001100110100110011001000110

Was hast du gelernt?

Mensch, das ist schon eine ganze Menge. In diesem Kapitel hast du

- Python installiert;
- gelernt, wie IDLE gestartet wird;
- den interaktiven Modus kennengelernt;
- Python Anweisungen gegeben, die es dann ausgeführt hat;
- gesehen, dass Python rechnen kann (sogar mit sehr großen Zahlen!);
- den IDLE-Texteditor gestartet, um dein erstes Programm einzugeben;
- dein erstes Python-Programm ausgeführt!
- gelernt, was Fehlermeldungen sind;
- dein zweites Python-Programm ausgeführt: Zahlen raten.

Teste dein Wissen

1. Wie startest du IDLE?
2. Was macht `print`?
3. Was ist das Multiplikationssymbol in Python?
4. Was zeigt IDLE an, wenn du ein Programm startest?
5. Welches andere Wort gibt es für „ein Programm laufen lassen“?

Probiere es aus

1. Lass Python im interaktiven Modus ausrechnen, wie viele Minuten eine Woche hat.
2. Schreibe ein kurzes Programm, das drei Zeilen ausgibt: Deinen Namen, dein Geburtsdatum und deine Lieblingsfarbe. Die Ausgabe sollte in etwa wie folgt aussehen:

```
Ich heiße Warren Sande.
Ich bin am 1. Januar 1970 geboren.
Meine Lieblingsfarbe ist blau.
```

Speichere das Programm und führe es aus. Wenn das Programm nicht das tut, was du willst, oder wenn du Fehlermeldungen bekommst, versuche, den Fehler zu beheben und es zum Laufen zu bringen.

Kapitel 16

Grafik

Du hast bereits eine Menge über die Grundelemente der Computerprogrammierung gelernt: Ein- und Ausgabe, Variablen, Entscheidungen, Schleifen, Listen, Funktionen, Objekte und Module. Ich hoffe, es hat dir Spaß gemacht, all diese Dinge in den Kopf zu bekommen. Jetzt wird es aber Zeit, etwas mehr Spaß mit Programmieren und Python zu haben.

In diesem Kapitel wirst du lernen, wie du Dinge auf den Bildschirm zeichnen kannst, zum Beispiel Linien, Formen, Farben und sogar kleine Animationen. Das wird uns in den nächsten Kapiteln helfen, Spiele und andere Programme zu schreiben.

Hilfe holen – Pygame

Es kann recht kompliziert sein, Grafik (und Sound) auf deinem Computer zum Laufen zu bringen. Daran sind das Betriebssystem, die Grafikkarte und viel Programmierarbeit beteiligt, um die wir uns eigentlich nicht kümmern wollen. Also verwenden wir ein Python-Modul namens Pygame, um es uns etwas einfacher zu machen.

Mit Pygame kannst du Grafik und auch alles andere erstellen, was du benötigst, damit Spiele auf verschiedenen Computern und Betriebssystemen funktionieren, ohne dich um die verworrenen Details jedes einzelnen Systems kümmern zu müssen. Pygame ist kostenlos, und eine Version von Pygame ist in diesem Buch enthalten. Wenn du Python mit dem zu diesem Buch gehörigen Installer installiert hast, ist Pygame bereits auf deinem Rechner vorhanden. Wenn nicht, musst du es separat installieren. Du kannst es von der Pygame-Website www.pygame.org herunterladen.

Ein Pygame-Fenster

Als Erstes müssen wir ein Fenster öffnen, in dem wir anfangen können, unsere Grafik zu zeichnen. Listing 16.1 zeigt ein ganz einfaches Programm, das ein Pygame-Fenster erstellt.

Listing 16.1 Ein Pygame-Fenster erstellen

```
import pygame
pygame.init()
screen = pygame.display.set_mode([640, 480])
```

Nun führe dieses Programm aus. Was siehst du? Abhängig von deinem Betriebssystem hast du vielleicht gesehen, wie auf dem Bildschirm ganz kurz ein Fenster (mit schwarzem Inhalt) eingeblendet wurde. Oder du bekommst ein Fenster, das sich nicht schließen lässt, wenn du es versuchst. Was soll das?

Nun, Pygame ist geschaffen, um Spiele zu programmieren. Und Spiele machen nicht nur ihr eigenes Ding, sondern müssen von einem Spieler bedient werden (mit ihm interagieren). Daher hat Pygame eine sogenannte *Ereignisschleife*, die dauernd prüft, ob der Benutzer etwas tut, etwa Tasten drückt oder die Maus bewegt oder das Fenster schließt. Pygame-Programme müssen die Ereignisschleife immer weiter laufen lassen; sobald die Schleife anhält, hält auch das Programm an. Da wir sie in unserem ersten Pygame-Programm nicht gestartet haben, hat das Programm nicht korrekt funktioniert.

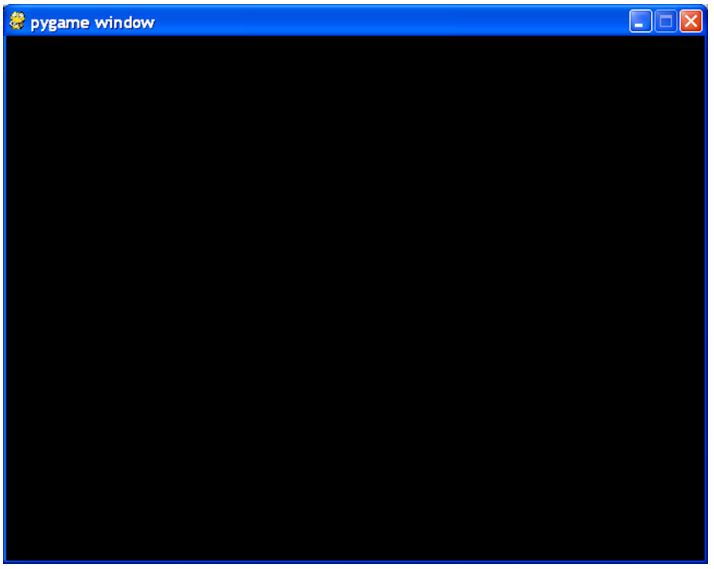
Du kannst die Pygame-Ereignisschleife mit einer **`while`**-Schleife am Laufen halten. Die Schleife soll solange laufen, wie der Benutzer unser Spiel spielt. Weil Pygame-Programm normalerweise nicht immer ein Menü haben, muss der Benutzer das Programm durch einen Klick auf das X in der oberen rechten Fensterecke (bei Windows) oder auf die Schließen-Schaltfläche oben links (bei Mac OS) schließen. Bei Linux-Systemen variiert das Symbol fürs Fenster schließen abhängig vom Window-Manager und dem verwendeten GUI-Framework, aber wenn du mit Linux arbeitest, nehme ich mal an, dass du weißt, wie man ein Fenster schließt!

Der Code im nächsten Listing öffnet ein Pygame-Fenster und hält es solange offen, bis der Benutzer es wieder schließt.

Listing 16.2 Das Pygame-Fenster korrekt arbeiten lassen

```
import pygame
pygame.init()
screen = pygame.display.set_mode([640, 480])
aktiv  = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

Starte das Listing, und dann solltest du ein Pygame-Fenster bekommen, das korrekt arbeitet und sich schließen lässt, wenn du das willst.



Wie genau funktioniert der Code in dieser `while`-Schleife? Sie verwendete die Pygame-*Ereignisschleife*. Aber dieses Thema verschieben wir auf Kapitel 18, wo es um Ereignisse gehen wird.

In das Fenster zeichnen

Nun haben wir ein Pygame-Fenster, das offen bleibt, bis wir es schließen – und das macht es dann auch sehr elegant! Das [640, 480] in der dritten Zeile von Listing 16.2 ist die Größe des Fensters: 640 Pixel breit und 480 Pixel hoch. Darin wollen wir nun ein paar Grafiken zeichnen. Ändere dein Programm, damit es wie Listing 16.3 aussieht.

Listing 16.3 Einen Kreis zeichnen

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
```

↓ Die folgenden drei Zeilen fügst du hinzu

```
screen.fill([255,255,255])   ← füllt das Fenster mit weißem Hintergrund
pygame.draw.circle(screen, [255,0,0],[100,100], 30, 0) ← zeichnet den Kreis
pygame.display.flip()   ← lässt den Computer ausflippen (ist nur ein Witz)
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

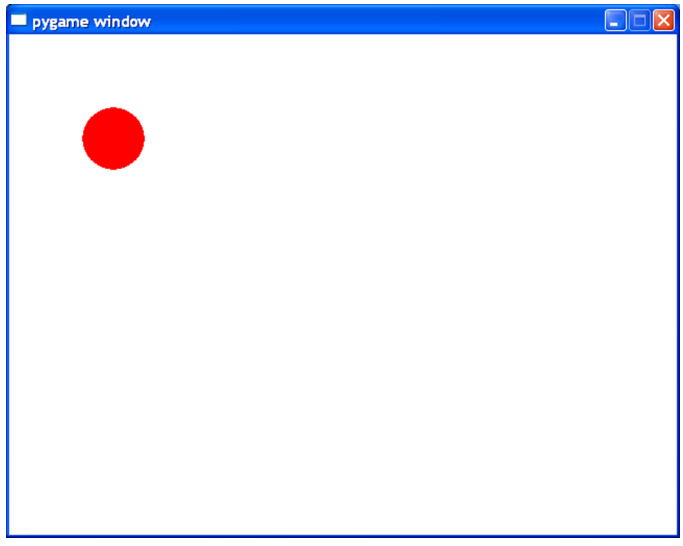
Was bedeutet das „flip“?

Das Anzeigeobjekt in Pygame (unseres heißt `screen` [englisch für „Bildschirm“] und wurde in Zeile 3 von Listing 16.3 angelegt) hat zwei Kopien von dem, was im Pygame-Fenster angezeigt wird. Der Grund: Wenn wir mit Animationen anfangen, möchten wir diese so schnell und reibungslos wie möglich anzeigen. Anstatt die Anzeige bei jeder kleinen Änderung an der Grafik direkt zu aktualisieren, wollen wir daher eine Reihe von Änderungen vornehmen und erst dann auf die neue Version der Grafik umstellen. „flip“ bewirkt diese Umstellung. So erscheinen die Änderungen alle gleichzeitig anstatt nacheinander. Dadurch erscheinen keine halbfertigen Kreise (oder Aliens oder was auch immer) auf unserem Bildschirm.

Die beiden Kopien sind der aktuelle und der nächste Bildschirm. Der aktuelle Bildschirm ist das, was du jetzt siehst, und der „nächste“ ist das, was wir sehen werden, wenn wir „flip“ sagen. Wir nehmen alle unsere Änderungen am nächsten Bildschirm vor und stellen dann mit flip auf diesen um, damit wir ihn sehen können.

Einen Kreis zeichnen

Wenn du das Programm in Listing 16.3 ausführst, müsstest du einen roten Kreis oben links im Fenster sehen:



Kein Wunder, die Funktion `pygame.draw.circle()` („draw“ ist englisch für „zeichnen“) zeichnet einen Kreis. Fünf Dinge musst du ihr angeben:

- Auf welcher *Oberfläche* der Kreis erscheinen soll. (Hier ist es die in Zeile 3 definierte Oberfläche namens `screen`. Das ist die Oberfläche der Anzeige.)

- In welcher *Farbe* er gezeichnet werden soll. (Hier ist die Farbe Rot und wird durch `[255, 0, 0]` dargestellt).
- An welcher *Position* er gezeichnet werden soll. (Hier `[100, 100]`, also von der oberen, linken Ecke 100 Pixel nach unten und 100 Pixel nach rechts.)
- In welcher *Größe* er gezeichnet werden soll. (Hier beträgt die Größe 30. Das ist der Radius in Pixeln, wobei der Radius die Entfernung vom Mittelpunkt des Kreises zu seinem Umfang ist.)
- Die *Linienstärke*. (Wenn diese Null ist, also `breite = 0`, dann ist der Kreis komplett mit Farbe gefüllt, wie es hier der Fall ist.)

Diese fünf Dinge werden wir uns jetzt genauer ansehen.

Worterklärung

Das Wort *Pixel* bedeutet „Bildpunkt“. Das ist ein Punkt auf deinem Bildschirm oder in einem Bild. Wenn du ein Bild mit einem Anzeigeprogramm stark vergrößerst, kannst du die einzelnen Pixel sehen. Links siehst Du eine normale Ansicht eines Fotos und rechts eine vergrößerte, in der du die Pixel erkennen kannst.

Mann, du hast aber gute Augen! Die kleinen Linien sind tatsächlich die Pixelzeilen. Ein typischer Computerbildschirm kann 768 Zeilen mit Pixeln haben, und in jeder Zeile sind 1.024 Pixel. Wir sagen dann, der Bildschirm hat eine Auflösung von 1.024 x 768. Manche Bildschirme haben mehr, andere weniger Pixel.

Oberflächen in Pygame

Wenn ich dich in der Wirklichkeit bitten würde, ein Bild zu zeichnen, würdest du fragen: „Worauf denn?“ In Pygame ist das, worauf wir zeichnen, eine *Oberfläche* (auf

Englisch „surface“). Die *Anzeigeoberfläche* ist die Bildschirmoberfläche. Wir nannten sie in Listing 16.3 `screen`. Doch ein Pygame-Programm kann viele Oberflächen haben, und du kannst Bilder von einer Oberfläche auf eine andere kopieren. Du kannst auch noch anderes mit einer Oberfläche tun, sie zum Beispiel drehen und skalieren (kleiner oder größer machen).

Wie gesagt existieren zwei Kopien der Anzeigeoberfläche. In der Sprache der Software sagen wir: Die Anzeigeoberfläche ist *doppelt gepuffert*. Das soll verhindern, dass wir halbfertige Formen und Bilder auf dem Bildschirm sehen. Wir zeichnen unsere Kreise, Aliens oder andere Dinge im Puffer und stellen dann diese Anzeigeoberfläche um (mit „flip“), um die fertig gezeichneten Bilder zu sehen.

Farben in Pygame

Das Farbsystem in Pygame wird auch von vielen anderen Computersprachen und Programmen verwendet. Es heißt *RGB*. R, G und B stehen für Rot, Grün und Blau.

Vielleicht hast du in Sachkunde schon gelernt, dass du aus den *Primärfarben* Rot, Grün und Blau jede Farbe mischen kannst. Genauso funktioniert das auch auf Computern. Jede Farbe (Rot, Grün und Blau) bekommt eine Zahl, und Farben werden als Liste von drei Integern angegeben, jede im Bereich zwischen 0 und 255. Wenn alle Nummern 0 sind, ist von keiner Farbe etwas vorhanden; also ist alles ganz dunkel, und du bekommst die Farbe Schwarz. Sind alle Farbwerte 255, bekommst du jeweils das Hellste von allen drei Farben, also Weiß. Wenn du so etwas wie [255, 0, 0] hast, dann bekommst du reines Rot, ohne Grün und Blau. Reines Grün wäre [0, 255, 0] und reines Blau [0, 0, 255]. Wenn alle drei Farben gleich sind wie in [150, 150, 150], bekommst du einen Grauton. Je kleiner die Zahlen, desto dunkler wird die Schattierung, und je höher die Zahlen, umso heller wird das Ganze.

Farbnamen

Pygame hat eine Liste von Farben, die du benutzen kannst, wenn du die Farbnotation mit [R, G, B] nicht anwenden möchtest. Es gibt über 600 definierte Farbnamen. Diese werde ich hier nicht alle auflisten, aber wenn du sie sehen möchtest, kannst du auf der Festplatte die Datei namens **colordict.py** suchen und in einem Editor öffnen.

Wenn du die englischen Farbnamen verwenden möchtest, musst du diese Zeile am Anfang deines Programms hinzufügen:

```
from pygame.color import THECOLORS
```

Möchtest du dann eine der benannten Farben verwenden, gehst du folgendermaßen vor (hier bezogen auf unser Kreisbeispiel):

```
pygame.draw.circle(screen, THECOLORS["red"],[100,100], 30, 0)
```

Wenn du mit Kombinationen von Rot, Grün und Blau etwas herumspielen und experimentieren möchtest, um verschiedene Farben zu mischen, kannst du das Programm **colormixer.py** ausprobieren, das vom Installer dieses Buches in den **\Beispiele**-Ordner geladen wurde. Damit kannst du jede Kombination von Rot, Grün und Blau erstellen.

Was passiert da drin?

Warum 255? Im Bereich von 0 bis 255 haben wir 256 verschiedene Werte für jede Primärfarbe (rot, grün und blau). Doch warum gerade diese Zahl, und nicht 200 oder 300 oder 400?

Mit 8 Bits kannst du genau 256 verschiedene Werte darstellen. Das sind alle möglichen Kombinationen von acht Nullen und Einsen. Acht Bits heißen auch ein Byte, und ein Byte ist der kleinste Speicherplatz, der eine eigene Adresse hat. Anhand einer Speicheradresse findet der Computer bestimmte Speicherplätze wieder.

Das ist wie in deiner Straße: Dein Haus oder deine Wohnung hat eine Adresse, aber dein Zimmer nicht. Ein Haus ist die kleinste „adressierbare Einheit“ in der Straße, und ein Byte ist die kleinste „adressierbare Einheit“ im Computerspeicher.

Man hätte auch mehr als 8 Bits für jede Farbe nehmen können, aber der nächstgrößere sinnvolle Wert wäre 16 Bits (2 Bytes), weil es reichlich unbequem ist, nur Teil-Bytes zu verwenden. Und es trifft sich nun einmal, dass 8 Bits ausreichen, um realistisch aussehende Farben zu erzeugen.

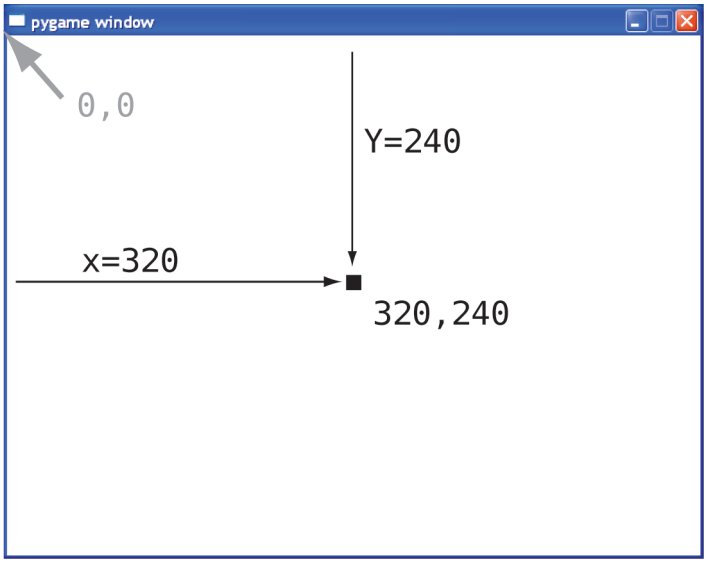
Da eine Farbe drei Werte (rot, grün, blau) mit je 8 Bits enthält, macht das insgesamt 24 Bits, sodass diese Art der Farbdarstellung auch „24-Bit-Farben“ genannt wird. Sie verwendet 24 Bits für jedes Pixel und 8 für jede Primärfarbe.

Positionen sind Bildschirmkoordinaten

Wenn wir etwas auf den Bildschirm zeichnen oder platzieren möchten, müssen wir angeben, wohin. Dazu gibt es zwei Zahlen: eine für die *x*-Achse (waagerechte oder horizontale Richtung) und eine für die *y*-Achse (senkrechte oder vertikale Richtung). In Pygame beginnen diese Zahlen in der oberen, linken Bildschirmecke mit [0, 0].

Tipp: Wenn du ein Zahlenpaar wie [320, 240] siehst, ist die erste Zahl die Horizontale (oder der Abstand vom linken Rand) und die zweite Zahl die Vertikale (oder der Abstand vom oberen Rand). In der Mathematik und in der Programmierung wird der Buchstabe x oft für eine horizontale und *y* für eine vertikale Strecke verwendet.

Wir haben unser Fenster 640 Pixel breit und 480 Pixel hoch gemacht. Wollten wir den Kreis in die Mitte setzen, müssten wir ihn an den Koordinaten [320, 240] zeichnen. Das sind 320 Pixel vom linken Rand nach rechts und 240 Pixel vom oberen Rand nach unten.



Wir wollen jetzt versuchen, einen Kreis in die Mitte des Fensters zu zeichnen. Hier ist das Programm.

Listing 16.4 Der Kreis rückt in die Fenstermitte

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
pygame.draw.circle(screen, [255,0,0],[320,240], 30, 0)
pygame.display.flip()
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

der dritte Parameter wurde von `[100, 100]` in `[320, 240]` umgeändert

Die Position [320, 240] wird als Mittelpunkt des Kreises verwendet. Vergleiche das Ergebnis von Listing 16.3 mit dem von Listing 16.4, dann siehst du den Unterschied.

Größe von Formen

Wenn du die **draw**-Funktionen von Pygame zum Zeichnen von Formen verwendest, musst du angeben, welche Größe die Form haben soll. Für einen Kreis gibt es nur eine einzige Größe: den Radius. Aber für so etwas wie ein Rechteck musst du die Breite und Höhe angeben.

Pygame hat ein spezielles Objekt namens **rect** (kurz für *Rechteck*), um rechtwinklige Flächen zu definieren. Ein **rect** wird mit den Koordinaten seiner oberen linken Ecke („links“ und „oben“) sowie seiner Breite („breite“) und Höhe („hoehe“) definiert:

```
Rect(links, oben, breite, hoehe)
```

Das definiert sowohl die Platzierung als auch die Größe. Hier ist ein Beispiel:

```
mein_rect = Rect(250, 150, 300, 200)
```

Das würde ein Rechteck erstellen, dessen obere, linke Ecke 250 Pixel vom linken Fensterrand und 150 Pixel vom oberen Fensterrand entfernt ist. Das Rechteck wäre 300 Pixel breit und 200 Pixel hoch. Probieren wir es aus.

Setze die folgende Zeile statt der Zeile 5 in Listing 16.4 ein und sieh, was geschieht:

```
pygame.draw.rect(screen, [255, 0, 0], [250, 150, 300, 200], 0)
```

Farbe des Rechtecks — Position und Größe des Rechtecks — Linienbreite (oder gefüllt)

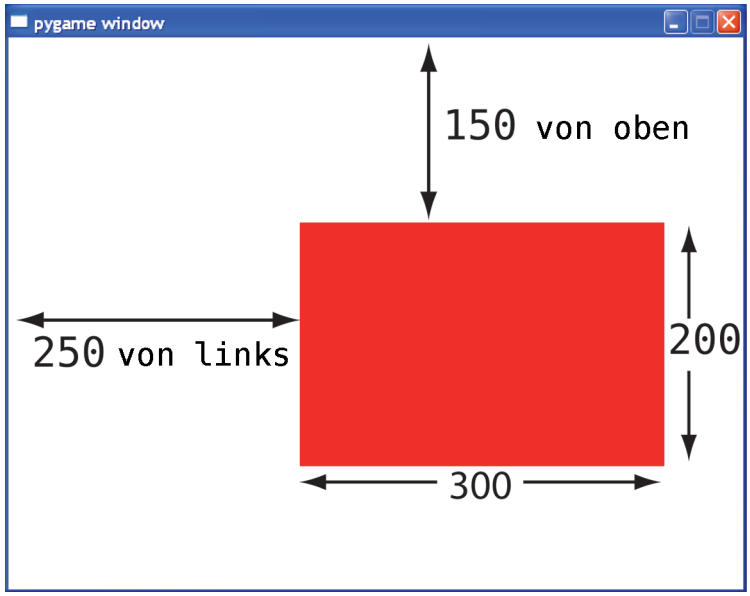
Die Position und Größe des Rechtecks kann eine einfache Liste (oder ein Tupel) mit Zahlen sein oder auch ein **Rect**-Objekt von Pygame. Du könntest also die obige Zeile auch durch zwei Zeilen wie diese ersetzen:

```
meine_liste = [250, 150, 300, 200]
pygame.draw.rect(screen, [255,0,0], meine_liste, 0)
```

oder

```
mein_rect = pygame.Rect(250, 150, 300, 200)
pygame.draw.rect(screen, [255,0,0], mein_rect, 0)
```

So müsste das Rechteck nun aussehen. Ich habe einige Maße angegeben, um zu zeigen, welche Zahlen welche Bedeutung haben.



Beachte, dass wir nur vier Argumente an `pygame.draw.rect` übergeben haben, da `rect` die Position und Größe in einem einzigen Argument unterbringt. In `pygame.draw.circle` sind Position und Größe zwei verschiedene Argumente, weshalb wir dieser Funktion fünf Argumente übergeben.

Denken wie ein (Pygame)-Programmierer

Wenn du ein Rechteck mit Rect(left, top, width, height) erstellst, kannst du auch noch weitere Attribute verwenden, um das Rect zu verschieben und auszurichten:

- **die vier Ränder:** `top, left, bottom, right`
- **die vier Ecken:** `topleft, bottomleft, topright, bottomright`
- **die Mitte jeder Seite:** `midtop, midleft, midbottom, midright`
- **der Mittelpunkt:** `center, centerx, centery`
- **die Maße:** `size, width, height`

Diese dienen nur der Bequemlichkeit. Wenn du also ein Rechteck so verschieben möchtest, dass sich sein Mittelpunkt an einem bestimmten Punkt befindet, musst du nicht ausrechnen, wo die oberen und unteren Koordinaten liegen müssen, sondern kannst direkt den Mittelpunkt bearbeiten.

Linienstärke

Um Formen zu zeichnen, müssen wir als Letztes die Strichbreite der Linie angeben. In unseren bisherigen Beispielen war das die 0, wodurch die gesamte Form ausgefüllt wird. Wenn wir eine andere Linienstärke verwenden würden, würden wir den Umriss der Figur sehen.

Ändere die Linienstärke (den letzten Parameter) einmal in 2:

```
pygame.draw.rect(screen, [255,0,0], [250, 150, 300, 200], 2)
```

Mach daraus eine 2

Probiere das aus und sieh, was geschieht. Nun versuche es auch mit anderen Linienstärken.

Moderne Kunst?

Wie wär's mit etwas computergenerierter moderner Kunst? Probiere doch spaßeshalber einmal den Code in Listing 16.5 aus. Du kannst den Code aus Listing 16.4 als Grundlage verwenden oder einfach von vorne anfangen.

Listing 16.5 Moderne Kunst mit `draw.rect`

```
import pygame, sys, random
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
for i in range (100):
    breite = random.randint(0, 250)
    hoehe = random.randint(0, 100)
    oben = random.randint(0, 400)
    links = random.randint(0, 500)
    pygame.draw.rect(screen, [0,0,0], [links, oben, breite, hoehe], 1)
pygame.display.flip()
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

Führe dies aus und schau, was du bekommst. Es sollte etwa so aussehen:

Verstehst du, wie das Programm funktioniert? Es zeichnet hundert Rechtecke mit zufälligen Größen und Positionen. Damit es noch „künstlerischer" aussieht, kannst du Farben hinzufügen und auch die Linienstärke vom Zufall bestimmen lassen, wie in Listing 16.7.

Listing 16.6 Moderne Kunst mit Farbe

```
import pygame, sys, random
from pygame.color import THECOLORS
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
for i in range (100):
    breite = random.randint(0, 250)
    hoehe = random.randint(0, 100)
    oben = random.randint(0, 400)
    links = random.randint(0, 500)
    farbname = random.choice(THECOLORS.keys())   <-- mach dir hierüber
    farbe = THECOLORS[farbname]                      noch keine Gedanken
    linienbreite = random.randint(1, 3)
    pygame.draw.rect(screen, farbe, [links, oben, breite, hoehe], linienbreite)
pygame.display.flip()
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

Wenn du das ausführst, bekommst du jedes Mal eine andere Ausgabe. Ist einmal eine richtig schöne dabei, kannst du ihr einen coolen Titel wie etwa „Die Stimme der Maschine“ geben und an eine Kunstgalerie verkaufen!

Einzelne Pixel

Manchmal wollen wir keinen Kreis und kein Rechteck zeichnen, sondern einzelne Punkte oder Pixel. Vielleicht möchten wir zum Beispiel einmal ein Matheprogramm schreiben, um eine Sinuskurve zu zeichnen.

Keine Bange, wenn du nicht weißt, was eine Sinuskurve ist. Für dieses Kapitel reicht es zu wissen, dass es einfach eine Wellenform ist.

Auch die Matheformeln in den nächsten Beispielen brauchen dich nicht zu kümmern. Gib sie einfach so ein, wie sie in den Listings stehen. Sie sind nur ein Mittel, um die Wellenform so hinzubekommen, dass sie schön in dein Pygame-Fenster passt.

Da so etwas wie eine `pygame.draw.sinewave()`-Methode nicht existiert, müssen wir diese Kurve aus einzelnen Punkten selbst zeichnen. Dazu kannst du zum Beispiel winzige Kreise oder Rechtecke verwenden, die nur ein oder zwei Pixel groß sind. Listing 16.7 zeigt, wie das mit Rechtecken aussehen würde.

Listing 16.7 Kurven aus vielen kleinen Rechtecken zeichnen

```
import pygame, sys
import math
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
for x in range(0, 640):
    y = int(math.sin(x/640.0 * 4 * math.pi) * 200 + 240)
    pygame.draw.rect(screen, [0,0,0],[x, y, 1, 1], 1)
pygame.display.flip()
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

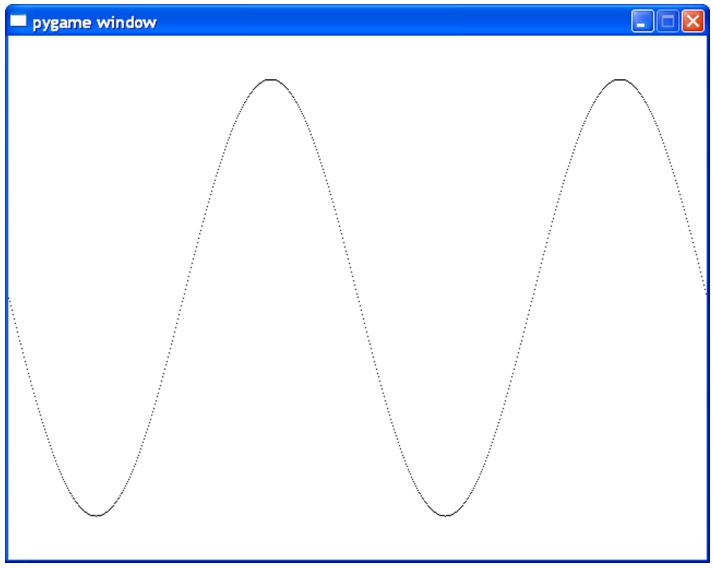
importiert Mathe-Funktionen, einschließlich `sin()`

Schleife von links nach rechts mit `x` = 0 bis 639

berechnet die `y`-Position jedes Punkts

zeichnet den Punkt als winziges Rechteck

Und so sieht das aus, wenn man es laufen lässt:



Zum Zeichnen jedes Punkts haben wir ein Rechteck verwendet, das 1 Pixel breit und 1 Pixel hoch ist. Beachte, dass wir eine Linienstärke von 1 und nicht 0 eingesetzt haben. Hätten wir 0 verwendet, wäre gar nichts angezeigt worden, weil es nichts auszufüllen gibt.

Die Punkte verbinden

Wenn du genau hinschaust, merkst du, dass die Sinuskurve nicht durchgehend ist; zwischen den Punkten sind kleine Lücken. Der Grund: Wenn die Kurve steil ansteigt oder abfällt, müssen wir um 3 Pixel nach oben (oder unten) gehen, wenn wir nur um ein Pixel nach rechts gehen. Und da wir keine Linien, sondern nur Punkte zeichnen, ist nichts da, um die Lücken auszufüllen.

Wir wollen jetzt die Punkte durch kleine Linien miteinander verbinden. Pygame hat eine Methode, um eine einzelne Linie zu zeichnen, und auch eine Methode, die Linien zwischen mehreren Punkten zeichnet (wie „Verbindungslinien"). Diese Methode ist **`pygame.draw.lines()`** und benötigt fünf Parameter:

- die Oberfläche **`surface`**, auf der sie zeichnen soll.
- eine Farbe **`color`**.
- eine Angabe, ob die Form geschlossen (**`closed`**) sein soll, indem eine Verbindungslinie vom letzten zurück zum ersten Punkt gezogen wird. Da wir unsere Sinuskurve nicht einschließen möchten, geben wir hier den Wert **`False`** an.
- eine Liste der zu verbindenden Punkte.
- Die Linienstärke.

In unserem Beispiel würde die **`pygame.draw.lines()`**-Methode daher wie folgt aussehen:

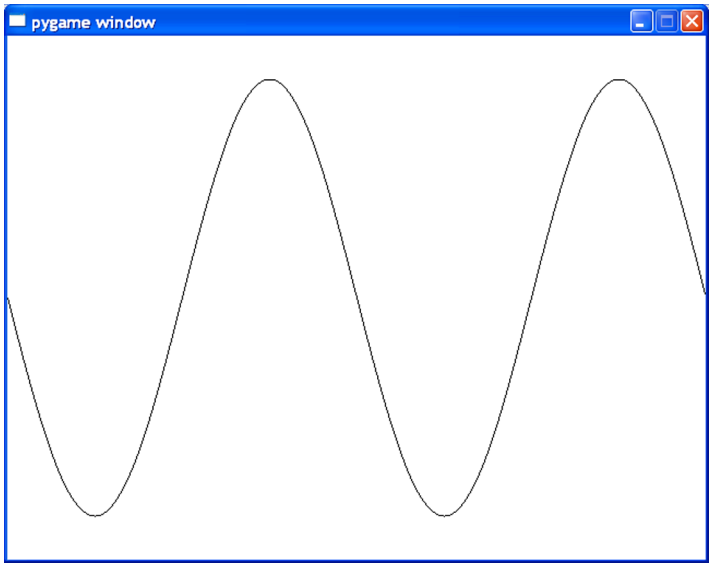
```
pygame.draw.lines(screen, [0,0,0],False, plotPunkte, 1)
```

In der **`for`**-Schleife zeichnen wir nicht jeden einzelnen Punkt, sondern erstellen einfach eine Liste von Punkten, die **`draw.lines()`** verbinden soll. Dann haben wir nur einen einzigen Aufruf an **`draw.lines()`**, der außerhalb der **`for`**-Schleife liegt. Das gesamte Programm wird nächsten Listing gezeigt.

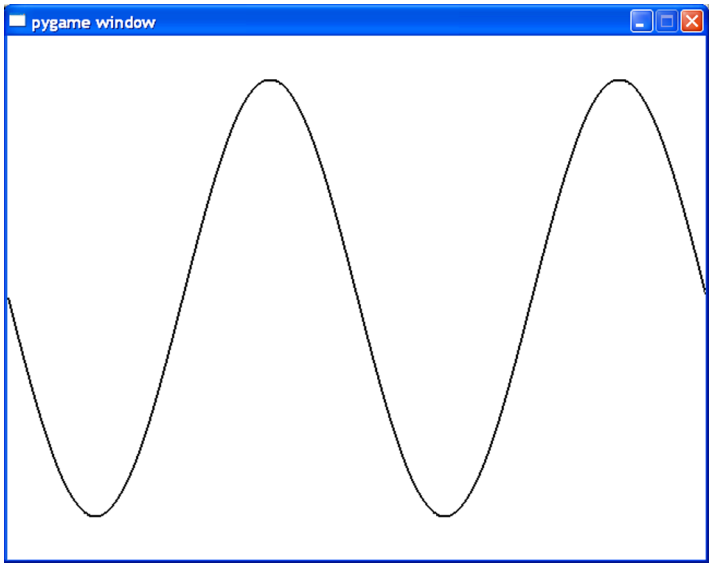
Listing 16.8 Eine durchgehende Sinuskurve

```
import pygame, sys
import math   <-- importiert Mathe-Funktionen, einschließlich sin()
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
plotPunkte = []
for x in range(0, 640):
    y = int(math.sin(x/640.0 * 4 * math.pi) * 200 + 240)   <-- berechnet die y-Position jedes Punkts
    plotPunkte.append([x, y])   <-- fügt jeden Punkt zur Liste hinzu
pygame.draw.lines(screen, [0,0,0],False, plotPunkte, 1)   <-- zeichnet die gesamte Kurve mit der Funktion draw.lines()
pygame.display.flip()
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

Wenn wir das jetzt ausführen, sieht es so aus:



Schon besser: Hier sind keine Lücken mehr zwischen den Punkten. Wenn wir die Linienstärke auf 2 erhöhen, wird es noch besser:



Punkte verbinden, die Zweite

Erinnerst du dich noch an die Zeichenspiele für kleine Kinder, in denen es darum ging, auf Papier Punkte zu verbinden und ein Bild herauszubekommen? Hier ist eine Pygame-Version.

Das Programm in Listing 16.9 zeichnet mithilfe der `draw.lines()`-Funktion und einer Punkteliste eine Form. Um das geheime Bild offenzulegen, gibst du das Programm in Listing 16.9 ein. Diesmal kannst du nicht mogeln! Dieses Programm liegt nicht im **\Beispiele**-Ordner; du musst es schon eintippen, wenn du das geheimnisvolle Bild sehen willst. Da aber die Eingabe so vieler Zahlen rasch langweilig wird, kannst du die `punkte`-Liste in einer Textdatei im **\Beispiele**-Ordner oder auf der Website finden.

Listing 16.9 Das geheimnisvolle Punkteverbinden

```
import pygame, sys
pygame.init()

punkte = [[221, 432], [225, 331], [133, 342], [141, 310],
        [51, 230],  [74, 217],  [58, 153],  [114, 164],
        [123, 135], [176, 190], [159, 77],  [193, 93],
        [230, 28],  [267, 93],  [301, 77],  [284, 190],
        [327, 135], [336, 164], [402, 153], [386, 217],
        [409, 230], [319, 310], [327, 342], [233, 331],
        [237, 432]]

screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
pygame.draw.lines(screen, [255,0,0],True, Punkte, 2)
pygame.display.flip()
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

← *dieses Mal* **closed=True**, *d. h. eine geschlossene Form*

Zeichnen, Punkt für Punkt

Gehen wir kurz noch einmal zurück zum Punkt-für-Punkt-Zeichnen. Es ist doch dumm, jedes Mal einen kleinen Kreis oder ein Rechteck zu zeichnen, wenn wir eigentlich nur die Farbe eines Pixels ändern möchten. Statt die `draw`-Funktionen einzusetzen, kannst du auch auf jedes einzelne Pixel auf der Oberfläche mit der `Surface.set_at()`-Methode zugreifen. Du sagst ihr, welches Pixel du auf welche Farbe einstellen möchtest:

```
screen.set_at([x, y], [0, 0, 0])
```

Wenn wir diese Codezeile in unser Sinuskurvenbeispiel einsetzen (anstelle von Zeile 8 in Listing 16.7), sieht die Kurve genauso aus, als hätten wir ein Pixel breite Rechtecke verwendet.

Um zu prüfen, auf welche Farbe ein Pixel bereits eingestellt ist, verwendest du die `Surface.get_at()`-Methode. Du übergibst ihr einfach die Koordinaten des zu prüfenden Pixels: `pixel_farbe = screen.get_at([320, 240])`. In diesem Beispiel war `screen` der Name der Oberfläche.

Bilder

Formen, Linien und einzelne Pixel auf den Bildschirm zu zeichnen, ist nur eine Spielart von Grafik. Manchmal möchten wir auch Bilder einsetzen, die wir von einer anderen Stelle bekommen, vielleicht aus einem Digitalfoto, aus heruntergeladenen Webinhalten oder etwas, das in einem Bildbearbeitungsprogramm erstellt wurde. In Pygame lassen sich Bilder am leichtesten mit den `image`-Funktionen benutzen.

Betrachten wir ein Beispiel. Wir werden ein Bild anzeigen, das bereits auf deiner Festplatte liegt, wenn du den Python-Installer von diesem Buch verwendet hast. Der Installer hat einen Unterordner namens **bilder** im **\Beispiele**-Ordner erstellt, und darin finden wir **beach_ball.png**, die Datei, die wir für dieses Beispiel brauchen. In Windows liegt die Datei zum Beispiel unter **c:\helloworld\Beispiele\bilder\beach_ball.png**.

Kopiere die Datei **beach_ball.png** an den Ort, wo du die Python-Programme speicherst, während du diese Beispiele durcharbeitest. So kann Python sie leicht wiederfinden, wenn das Programm läuft. Wenn **beach_ball.png** im richtigen Verzeichnis liegt, gib das Programm aus Listing 16.10 ein und probiere es aus.

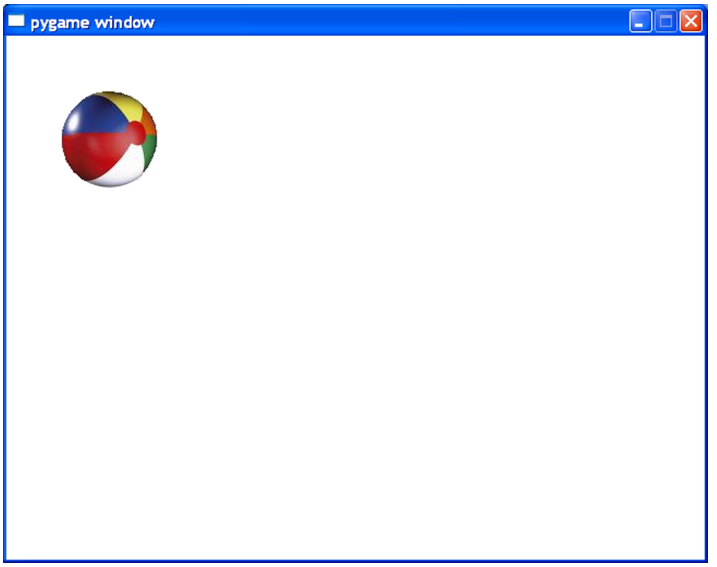
Wenn du nicht den Installer zum Buch verwendet hast, kannst du beach_ball.png von der beiliegenden CD herunterladen.

Listing 16.10 Ein Wasserball-Bild in einem Pygame-Fenster anzeigen

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
mein_ball = pygame.image.load("beach_ball.png")
screen.blit(mein_ball, [50, 50])
pygame.display.flip()
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

neu sind nur diese beiden Zeilen

Wenn du dieses Programm ausführst, siehst du das Bild eines Wasserballs nahe der oberen, linken Ecke des Pygame-Fensters:



In Listing 16.10 sind nur die Zeilen 5 und 6 neu; alles andere hast du bereits in den Listings 16.3 bis 16.9 gesehen. Wir haben den `draw`-Code der früheren Beispiele durch Code ersetzt, der ein Bild von der Festplatte lädt und anzeigt.

In Zeile 5 lädt die Funktion `pygame.image.load()` das Bild von der Festplatte und erstellt ein Objekt namens `mein_ball`. Dieses `mein_ball`-Objekt ist eine Oberfläche. (Oberflächen haben wir einige Seiten zuvor besprochen.) Doch diese Oberfläche können wir nicht sehen; sie existiert nur im Speicher. Die einzige Oberfläche, die wir sehen, ist die der *Anzeige*, die sogenannte `screen`. (Wir erstellten sie in Zeile 3.) Zeile 6 kopiert die Oberfläche `mein_ball` auf die Oberfläche `screen`. Dann macht `display.flip()` sie sichtbar, wie du es bereits kennst.

Schon gut, Carter, schon sehr bald werden wir anfangen, den Ball rollen zu lassen.

Hast du das komische Ding in Zeile 6 von Listing 16.10 bemerkt, dieses `screen.blit()`? Was bedeutet `blit`? Schau in die Worterklärung, um es herauszufinden.

Worterklärung

In der Grafikprogrammierung werden sehr oft Pixel von einem Ort zum anderen kopiert (z.B. von einer Variablen auf dem Bildschirm oder von einer Oberfläche auf eine andere). Das Kopieren von Pixeln nennt man in der Programmierung auch *Blitting* (ein englisches Wort).

In Pygame kopieren wir *blit*-Pixel von einer *Oberfläche* auf eine andere. Hier haben wir die Pixel von der Oberfläche `mein_ball` auf die Oberfläche `screen` kopiert.

In Zeile 6 von Listing 16.10 haben wir das Bild des Wasserballs an die Position 50, 50 *kopiert*. Das bedeutet: jeweils 50 Pixel vom linken und vom oberen Rand des Fensters entfernt. Wenn du mit einer `surface`-Oberfläche oder einem `rect`-Rechteck arbeitest, wird damit die obere linke Ecke des Bilds platziert. Also ist der obere, linke Punkt des Wasserballs jeweils 50 Pixel vom linken und vom oberen Rand des Fensters entfernt.

Jetzt kommt Bewegung in die Sache!

Da wir nun Grafiken auf unserem Pygame-Fenster darstellen können, wollen wir diese auch in Bewegung bringen. Das nennt man Animation! Computeranimation bedeutet eigentlich nur, Bilder (Pixel-Gruppen) von einem Ort zum anderen zu bewegen. Das tun wir jetzt mit unserem Wasserball.

Um ihn zu bewegen, müssen wir seine Position ändern. Zuerst versuchen wir, ihn zur Seite zu bewegen. Um sicherzustellen, dass die Bewegung auch wahrnehmbar ist, verlagern wir ihn gleich 100 Pixel nach rechts. Die Links-Rechts-Richtung (horizontal) ist die erste Zahl in dem Zahlenpaar, das die Position angibt. Um etwas um 100 Pixel nach rechts zu verschieben, müssen wir also die erste Zahl um 100 erhöhen. Außerdem bauen wir eine Verzögerung ein, damit wir sehen können, wie die Animation verläuft.

Ändere das Programm von Listing 16.10 nun wie in Listing 16.11 um. (Du musst die Zeilen 8, 9 und 10 vor der `while`-Schleife einfügen.)

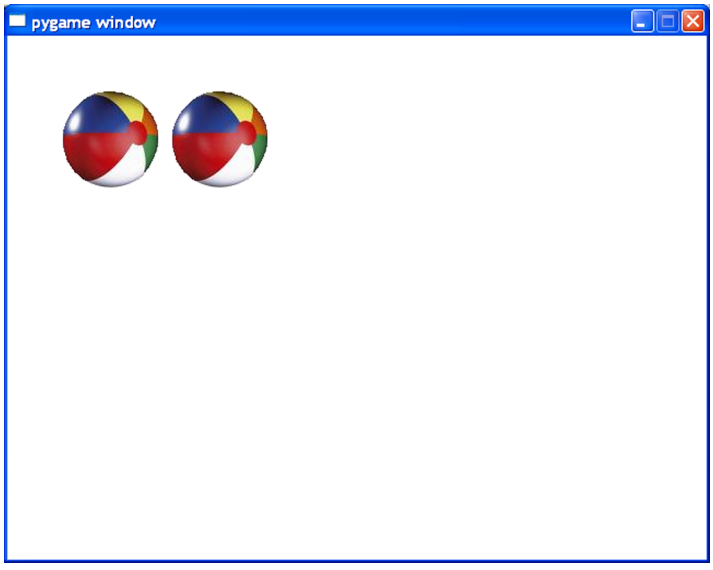
Listing 16.11 Wir versuchen, den Wasserball zu bewegen

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
mein_ball = pygame.image.load('beach_ball.png')
screen.blit(mein_ball,[50, 50])
pygame.display.flip()
pygame.time.delay(2000)
screen.blit(mein_ball, [150, 50])
pygame.display.flip()
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

das sind die drei neuen Zeilen

Führe das Programm aus und sieh, was passiert. Hat sich der Ball bewegt? Ja, gewissermaßen … Es waren zwei Wasserbälle zu sehen:

Der erste erschien an der ursprünglichen Position und der zweite nach einigen Sekunden rechts davon. Wir haben also den Wasserball nach rechts verschoben, aber dabei eines vergessen: den ersten Ball zu *löschen*!



Animation

In der Computeranimation besteht eine Bewegung immer aus zwei Schritten:

1. Wir zeichnen eine Sache an der neuen Position.
2. Wir löschen die Sache von der alten Position.

Den ersten Teil haben wir bereits gesehen: Wir haben den Ball an einer neuen Position gezeichnet. Jetzt müssen wir ihn auch noch von seinem Ursprungsort löschen. Aber was bedeutet „löschen“ eigentlich?

Bilder löschen

Wenn du etwas auf Papier oder eine Tafel zeichnest, ist es leicht zu löschen. Du nimmst dazu einfach einen Radiergummi oder Schwamm. Aber was tust du mit einer Malerei? Angenommen, du hast einen blauen Himmel gezeichnet und dann einen Vogel, der darin fliegt. Wie kannst du den Vogel „löschen"? Malerfarben kann man nicht löschen. Du musst die Stelle, wo der Vogel flog, wieder mit neuem blauem Himmel übermalen.

Computergrafiken verhalten sich in dieser Hinsicht wie Malerei und nicht wie Bleistift oder Kreide. Um etwas zu „löschen", malst du in Wirklichkeit darüber. Und womit? Im Falle des blauen Himmels würdest du den Vogel übermalen, und da der Himmel blau ist, nimmst du blaue Farbe dafür. Da in unserem Fall der Hintergrund weiß ist, müssen wir das Originalbild des Wasserballs weiß übermalen.

Versuchen wir das. Ändere dein Programm aus Listing 16.11 in Listing 16.12 um. Du musst nur eine einzige Zeile hinzufügen.

Listing 16.12 Nächster Versuch, einen Wasserball zu bewegen

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
mein_ball = pygame.image.load('beach_ball.png')
screen.blit(mein_ball,[50, 50])
pygame.display.flip()
pygame.time.delay(2000)
screen.blit(mein_ball, [150, 50])
pygame.draw.rect(screen, [255,255,255], [50, 50, 90, 90], 0)
pygame.display.flip()
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

diese Zeile löscht den alten Ball

Wir fügten Zeile 10 hinzu, um den ersten Wasserball mit einem weißen Rechteck zu übermalen. Das Ball-Bild ist etwa 90 Pixel breit und 90 Pixel hoch, also machen wir das Rechteck genauso groß. Wenn du das Programm in Listing 16.12 ausführst, sollte es so aussehen, als sei der Wasserball von seinem Ursprungsort an die neue Position gesprungen.

Was liegt darunter?

Etwas mit einem weißen Hintergrund oder einem blauen Himmel zu übermalen ist ziemlich einfach. Aber was tun wir, wenn der Vogel in einem wolkigen Himmel fliegt? Oder wenn Bäume im Hintergrund stehen? Dann müssten wir den Vogel mit Wolken oder Bäumen übermalen, um ihn zu löschen. Das Wichtige daran ist, dass du genau

wissen musst, wie der Hintergrund aussieht, also was „unter“ deinen Bildern liegt, denn wenn du sie verschiebst, musst du alles wieder herstellen wie zuvor.

In unserem Beispiel mit dem Wasserball ist das einfach, weil der Hintergrund weiß ist. Wäre er jedoch eine Strandszene, so würde es kniffliger. Wir müssten dann den richtigen Teil aus dem Hintergrundbild statt nur weiße Farbe zum Übermalen verwenden. Oder wir müssten die gesamte Szene neu malen und den Wasserball dabei an seine neue Position versetzen.

Weichere Animationen

Bisher ließen wir unseren Ball nur eine einzige Bewegung vollziehen. Mal sehen, ob wir ihn auch etwas realistischer in Gang setzen können. Wenn wir Sachen auf dem Bildschirm animieren, empfiehlt es sich, die Bewegung in kleinen Schritten zu vollziehen, damit sie weicher aussieht. Wir wollen also versuchen, unseren Ball schrittweise zu bewegen.

Dabei werden wir die Schritte nicht nur kleiner machen, sondern eine Schleife hinzufügen, um den Ball zu bewegen (da wir schließlich eine ganze Menge kleiner Schritte ausführen wollen). Bearbeite den Code von Listing 16.12, bis er so aussieht wie Listing 16.13.

Listing 16.13 Den Wasserball geschmeidig bewegen

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
mein_ball = pygame.image.load('beach_ball.png')
x = 50
y = 50
screen.blit(mein_ball,[x, y])
pygame.display.flip()
for zaehler in range (1, 100):
    pygame.time.delay(20)
    pygame.draw.rect(screen, [255,255,255], [x, y, 90, 90], 0)
    x = x + 5
    screen.blit(mein_ball,[x, y])
    pygame.display.flip()

aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
pygame.quit()
```

diese beiden Zeilen hinzufügen

verwendet x und y (statt Zahlen)

startet eine for-Schleife

Verzögerung wird von 2000 auf 20 heruntergesetzt

verwendet x und y (statt Zahlen)

Wenn du dieses Programm ausführst, siehst du, wie sich der Ball von seiner Originalposition zur rechten Seite des Fensters bewegt.

Den Ball in Bewegung halten

Im vorigen Programm bewegte sich der Ball auf die rechte Seite des Fensters und blieb da liegen. Jetzt wollen wir ihn in Bewegung halten.

Was würde geschehen, wenn wir **x** einfach immer weiter erhöhen? Der Ball würde immer weiter nach rechts rollen. Aber unser Fenster (die Anzeigeoberfläche) hört bei **x = 640** auf. Also würde der Ball einfach verschwinden. Ändere einmal die **for**-Schleife aus Zeile 10 von Listing 16.13 folgendermaßen um:

```
    for zaehler in range (1, 200):
```

Da die Schleife jetzt doppelt so lange läuft, verschwindet der Ball hinterm Fensterrand! Wenn wir den Ball weiter sehen möchten, haben wir zwei Möglichkeiten:

- Wir lassen den Ball vom Fensterrand *abprallen*.
- Wir lassen den Ball auf der anderen Seite wieder *auftauchen*.

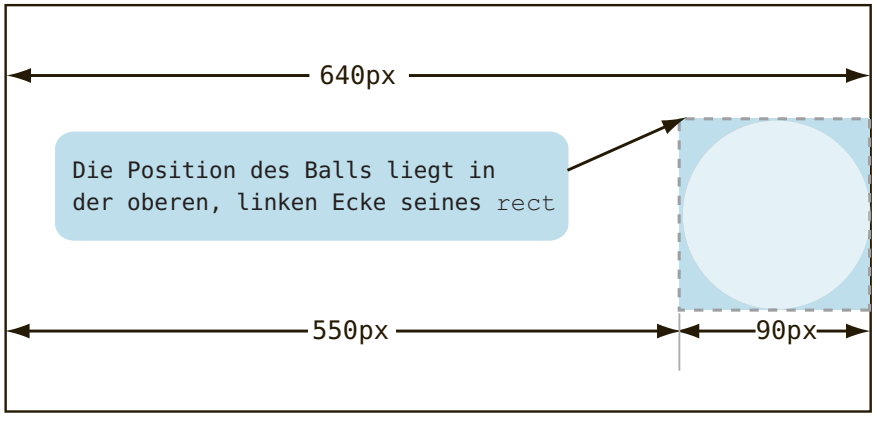
Wir wollen beides versuchen, um zu sehen, wie es funktioniert.

Den Ball abprallen lassen

Wenn es so aussehen soll, als würde der Ball vom Fensterrand *abprallen*, müssen wir wissen, wann er dort auftrifft, und ihn dann in die Gegenrichtung umlenken. Soll er immer hin und her springen, müssen wir dasselbe auch auf der rechten Seite des Fensters tun.

Am linken Rand ist das einfach, da wir nur schauen müssen, wann die Position des Balls 0 (oder eine andere kleine Zahl) ist.

Am rechten Rand müssen wir prüfen, wann der rechte Rand des Balls das Fenster berührt. Doch die Position des Balls wird von links (der oberen linken Ecke) an bestimmt und nicht von rechts. Also müssen wir die Breite des Balls subtrahieren.



Wenn der Ball zur rechten Fensterseite geht, müssen wir ihn an der Position 550 umlenken.

Um das zu vereinfachen, nehmen wir am Code einige Änderungen vor:

- Wir wollen den Ball immer hin und her springen lassen (oder zumindest solange, bis wir das Pygame-Fenster schließen). Da wir bereits eine **`while`**-Schleife haben, die läuft, solange das Fenster offen ist, können wir unseren Code zur Anzeige des Balls in diese Schleife verlagern. (Es handelt sich um die **`while`**-Schleife ganz unten in deinem Programm.)
- Anstatt immer 5 zur Ballposition zu addieren, legen wir die neue Variable **`x_geschwindigkeit`** an, um festzulegen, wie schnell sich der Ball bei jeder Iteration bewegen soll. Außerdem werde ich den Ball etwas schneller machen, indem ich den Wert auf 10 erhöhe.

Der neue Code steht in Listing 16.14.

Listing 16.14 Einen Wasserball abprallen lassen

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
mein_ball = pygame.image.load('beach_ball.png')
x = 50
y = 50
x_geschwindigkeit = 10   ← die x_geschwindigkeit-Variable

aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
                              hier, in der while-Schleife,
    pygame.time.delay(20)     steht der Code zum Anzeigen des Balls
    pygame.draw.rect(screen, [255,255,255], [x, y, 90, 90], 0)
    x = x + x_geschwindigkeit   ← die x_geschwindigkeit-Variable
    if x > screen.get_width() - 90  or  x < 0:
        x_geschwindigkeit = - x_geschwindigkeit   ← Wenn der Ball abprallt,
    screen.blit(mein_ball,[x, y])                   kehre die Richtung (Vorzeichen
    pygame.display.flip()                           von x_geschwindigkeit) um
pygame.quit()
```

Die Zeilen 19 und 20 sind es, die den Ball von den Fensterrändern abprallen lassen. In Zeile19 (**`if x > screen.get_width() - 90 or x < 0:`**) erkennen wir, ob der Ball am Fensterrand angekommen ist. Wenn ja, kehren wir in Zeile 20 (**`x_geschwindigkeit = - x_geschwindigkeit`**) seine Richtung um.

Probiere es und schau, wie es funktioniert.

Ballspiel in 2-D

Bisher haben wir den Ball nur in einer eindimensionalen Bewegung hin und her gerollt. Jetzt wollen wir ihn zugleich nach oben und unten bewegen. Dazu braucht es nur einige wenige Änderungen, wie Listing 16.15 zeigt.

Listing 16.15 Ballspiel in 2-D

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
mein_ball = pygame.image.load('beach_ball.png')
x = 50
y = 50
x_geschwindigkeit = 10
y_geschwindigkeit = 10   ← Code für y-Geschwindigkeit (vertikale Bewegung)
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
    pygame.time.delay(20)
    pygame.draw.rect(screen, [255,255,255], [x, y, 90, 90], 0)
    x = x + x_geschwindigkeit
    y = y + y_geschwindigkeit   ← Code für y-Geschwindigkeit (vertikale Bewegung)
    if x > screen.get_width() - 90  or  x < 0:
        x_geschwindigkeit = - x_geschwindigkeit
    if y > screen.get_height() - 90 or y < 0:      | Ball prallt oben
        y_geschwindigkeit = -y_geschwindigkeit     | oder unten ab
    screen.blit(mein_ball,[x, y])
    pygame.display.flip()
pygame.quit()
```

Wir haben dem vorigen Programm die Zeilen 9 (`y_geschwindigkeit = 10`), 18 (`y = y + y_geschwindigkeit`), 21 (`if y > screen.get_height() - 90 or y < 0:`) und 22 (`y_geschwindigkeit = -y_geschwindigkeit`) hinzugefügt. Probiere es noch einmal aus!

Möchtest du den Ball verlangsamen, so hast du auch hierzu mehrere Möglichkeiten:

- Du kannst die Variablen für die Geschwindigkeit heruntersetzen (`x_geschwindigkeit` und `y_geschwindigkeit`). Damit bewegt sich der Ball bei jedem Animationsschritt nicht so weit, und die Bewegung wird weicher.
- Du könntest die Verzögerung heraufsetzen. In Listing 16.15 beträgt sie 20. Dieser Wert ist in Millisekunden angegeben; das sind Tausendstel Sekunden. Also wartet das Programm bei jedem Schleifendurchlauf 0,02 Sekunden ab. Wenn du diese Zahl erhöhst, verlangsamst du die Bewegung, und wenn du sie verringerst, beschleunigst du die Bewegung.

Spiele ruhig etwas mit Geschwindigkeit und Verzögerung herum, um die Auswirkungen zu verstehen.

Den Ball umbrechen

Schauen wir uns nun die zweite Möglichkeit an, unseren Ball in Bewegung zu halten: Anstatt ihn vom Fensterrand abprallen zu lassen, werden wir ihn nach rechts heraus und von links wieder hineinrollen lassen. Das nennt man *Umbruch*. Das bedeutet, wenn der Ball auf der rechten Seite des Bildschirms verschwindet, erscheint er auf der linken Seite des Bildes wieder.

Der Einfachheit halber bewegen wir den Ball nur waagerecht (horizontal). Der neue Code steht in Listing 16.16.

Listing 16.16 Den Wasserball umbrechen

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
mein_ball = pygame.image.load('beach_ball.png')
x = 50
y = 50
x_geschwindigkeit = 5
aktiv = True
while aktiv:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            aktiv = False
    pygame.time.delay(20)
    pygame.draw.rect(screen, [255,255,255], [x, y, 90, 90], 0)
    x = x + x_geschwindigkeit
    if x > screen.get_width():   <-- Ist der Ball ganz rechts ...
        x = 0                    <-- ...geht's an der linken Seite wieder los.
    screen.blit(mein_ball,[x, y])
    pygame.display.flip()
pygame.quit()
```

In den Zeilen 17 (`if x > screen.get_width():`) und 18 (`x = 0`) erkannten wir, wann der Ball am rechten Fensterrand ankommt, und schickten ihn zurück zur linken Seite.

Vielleicht hast du gemerkt, dass der Ball bei [0, 50] eingeblendet wird, wenn er rechts erscheint. Es sähe natürlicher aus, wenn er in den Bildschirm hineingleiten würde. Ändere Zeile 18 (`x = 0`) in `x = -90` und schau, ob du einen Unterschied bemerkst.

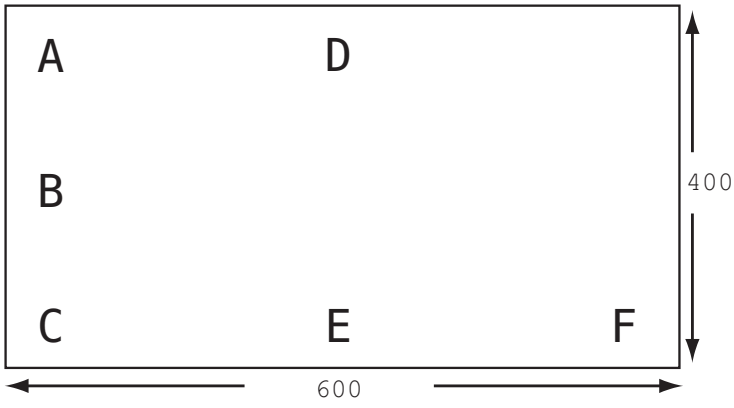
Was hast du gelernt?

Puh! Mannomann, in diesem Kapitel war eine Menge los! Du hast gelernt

- wie Pygame verwendet wird.
- wie du ein Grafikfenster erstellst und darin Formen zeichnest.
- wie Farben in einer Computergrafik eingestellt werden.
- wie du Bilder in ein Grafikfenster kopieren kannst.
- wie du Bilder animierst und außerdem „löschst“, wenn du sie an einen neuen Platz verlagerst.
- wie du einen Wasserball im Fenster hin und her springen lässt.
- wie du einen Wasserball aus dem Fenster hinaus und auf der anderen Seite wieder hinein rollen lassen kannst.

Teste dein Wissen

1 Welche Farbe erzeugt der RGB-Wert [255, 255, 255]?
2 Welche Farbe erzeugt der RGB-Wert [0, 255, 0]?
3 Mit welcher Pygame-Methode kannst du Rechtecke zeichnen?
4 Mit welcher Pygame-Methode kannst du Verbindungslinien zwischen Punkten zeichnen?
5 Was sind *Pixel*?
6 Wo liegt die Position [0, 0] in einem Pygame-Fenster?
7 Wenn ein Pygame-Fenster 600 Pixel breit und 400 Pixel hoch ist, welcher Buchstabe im unten stehenden Diagramm liegt dann an Position [50, 200]?



8 Welcher Buchstabe im unten stehenden Diagramm liegt an Position [300, 50]?
9 Mit welcher Pygame-Methode kopierst du Bilder auf eine Oberfläche (wie z. B. die Anzeigeoberfläche)?
10 Welche Schritte sind am wichtigsten, wenn du ein Bild „bewegst” oder animierst?

Probiere es aus

1 Wir haben besprochen, wie man Kreise und Rechtecke zeichnet. Pygame hat auch Methoden zum Zeichnen von Linien, Bögen, Ellipsen und Vielecken. Probiere sie aus, um in einem Programm verschiedene Formen zu zeichnen.
Mehr über diese Methoden findest du in der (englischen) Pygame-Dokumentation unter www.pygame.org/docs/ref/draw.html. Wenn du keinen Internetzugang hast, kannst du sie auch auf deiner Festplatte finden (sie wird zusammen mit Pygame installiert). Durchsuche deine Festplatte nach einer Datei namens **pygame_draw.html**.
Außerdem kannst du Pythons Hilfesystem verwenden (das wir am Ende von Kapitel 6 besprochen haben). SPE hat leider keine funktionierende interaktive Shell, also musst du IDLE starten und Folgendes eingeben:

```
>>> import pygame
>>> help()
help> pygame.draw
```

Du bekommst dann eine Liste von Zeichenmethoden mit Erklärungen dazu.

2 Versuche, eines der Beispielprogramme so zu ändern, dass es statt des Wasserballs ein anderes Bild anzeigt. Du kannst Beispielbilder aus dem Ordner **\Beispiele\images** verwenden oder eines herunterladen oder selbst zeichnen. Oder du verwendest ein Stück eines Digitalfotos.

3 Versuche, die Werte **`x_speed`** und **`y_speed`** in Listing 16.15 oder 16.16 so zu ändern, dass sich der Ball schneller und in verschiedene Richtungen bewegt.

4 Versuche, Listing 16.15 so zu ändern, dass der Ball statt vom Fensterrand von einer unsichtbaren Wand oder einem unsichtbaren Boden „abprallt".

5 Versuche, in den Listings 16.5 bis 16.9 (Moderne Kunst, Sinuskurve und Geheimnisvolles Punkteverbinden) die Zeile **`pygame.display.flip`** ins Innere der **`while`**-Schleife zu versetzen. Dazu musst du sie nur um vier Stellen einrücken. Nach dieser Zeile und innerhalb der **`while`**-Schleife fügst du mit dieser Zeile eine Verzögerung ein und schaust, was dann passiert:

```
pygame.time.delay(30)
```