Lerne Python, indem du Spiele entwickels

Alexander (Sasha) Pastukhov

2023-10-06

Contents

Chapter 1

Einleitung

Dieses Buch wird dich das Programmieren lehren. Hoffentlich auf eine spaßige Art und Weise, denn wenn es etwas Befriedigenderes gibt als ein Videospiel zu spielen, dann ist es, eines zu erstellen. Obwohl es für den Kurs *"Python für soziale und experimentelle Psychologie"* geschrieben wurde, ist mein Hauptziel nicht, dir Python an sich beizubringen. Python ist ein fantastisches Werkzeug (mehr dazu später), aber es ist nur eine von vielen existierenden Programmiersprachen. Mein ultimatives Ziel ist es, dir dabei zu helfen, allgemeine Programmierfähigkeiten zu entwickeln, die nicht von einer spezifischen Programmiersprache abhängen, und sicherzustellen, dass du gute Gewohnheiten entwickelst, die deinen Code klar, leicht lesbar und leicht zu warten machen. Dieser letzte Teil ist entscheidend. Programmieren geht nicht nur darum, Code zu schreiben, der funktioniert. Das muss natürlich gegeben sein, ist aber nur die Mindestanforderung. Programmieren geht darum, einen klaren und leicht lesbaren Code zu schreiben, den andere und, noch wichtiger, du selbst zwei Wochen später verstehen kannst.

1.1 Ziel des Buches

Das Ziel ist, dass du in der Lage bist, ein anspruchsvolles Experiment zu programmieren, das mehrere Blöcke und Versuche, verschiedene Bedingungen, komplizierte visuelle Darstellungen, automatische Datenaufzeichnung und Fehlerbehandlung haben kann. Wenn das ambitioniert klingt, dann ist es das auch, aber wir werden in kleinen Schritten vorgehen und im Prozess wirst du lernen.

- Kernkonzepte von Python einschließlich
  - Variablen und Konstanten
  - unveränderliche Datentypen wie Ganzzahlen, Fließkommazahlen, Zeichenketten, logische Werte und Tupel
  - veränderliche Typen wie Listen und Dictionaries

  - Funktionen
  - Steuerungsstrukturen wie if-else-Bedingungen und Schleifen
  - objektorientierte Programmierung einschließlich Konzepte des Vererbung, (optional) Duck-Typing und Mischung
  - Ausnahmen
  - Dateioperationen
- PsychoPy: Dies ist nicht Kern-Python, aber es ist eine erstklassige Bibliothek für psychophysische Experimente und du wirst die Schlüsselwerkzeuge lernen, die für die Programmierung eines Experiments erforderlich sind.
- Guter Programmierstil einschließlich
  - Klaren Code in iterativer Weise schreiben
  - Lesen (deines eigenen) Codes
  - Dokumentieren deines Codes
  - Debuggen deines Programms in VS Code

Ich habe versucht, Konzepte in dem Kontext zu präsentieren, der sie erfordert und daher am besten erklärt und ihre typischen Anwendungsszenarien darstellt. Das bedeutet, dass das Material rund um verteilt ist und auf Bedarfsbasis präsentiert wird. Zum Beispiel wird das Konzept der Listen zuerst präsentiert, aber Operationen daran werden in einem späteren Kapitel vorgestellt, sowohl weil wir dies erst später benötigen als auch um dein Gefühl, überwältigt zu sein, in einem vernünftigen Rahmen zu halten. Dies macht es schwieriger, dieses Buch als Referenz zu verwenden (es gibt ausgezeichnete Referenzen da draußen, angefangen mit der offiziellen Python-Dokumentation), aber die Hoffnung ist, dass es dir durch die Bereitstellung von mundgerechten Informationsbrocken leichter fallen wird, das Material zu verstehen und es mit dem zu integrieren, was du bereits weißt.

Die gleiche "verteilte" Logik gilt für Übungen. Anstatt am Ende jedes Kapitels aufgelistet zu sein, sind sie in den Text eingebettet und du solltest sie zu diesem Zeitpunkt erledigen. Viele von ihnen dienen dazu, Konzepte zu verdeutlichen, die zuvor anhand von anschaulichen Beispielen präsentiert wurden, daher wäre es hilfreich, sie sofort durchzuführen. Gleiches gilt für die Programmierübung, obwohl du in diesem Fall das gesamte Material lesen kannst, um eine "Vogelperspektive" auf das gesamte Programm zu bekommen und dann den Text erneut zu lesen und die Programmierübung durchzuführen.

1.2 Voraussetzungen

Dieses Material setzt keine Vorkenntnisse in Python oder Programmierung beim Leser voraus. Sein Ziel ist es, dein Wissen schrittweise aufzubauen und dir zu ermöglichen, immer komplexere Spiele zu erstellen.

1.3 Warum Spiele?

Der eigentliche Zweck dieses Kurses ist es, Psychologie- und Sozialwissenschaftsstudenten beizubringen, wie man *Experimente* programmiert. Das ist es, worum es in der echten Forschung geht. Allerdings gibt es zwischen den beiden praktisch wenig Unterschied. Die grundlegenden Zutaten sind die gleichen und, man könnte sagen, Experimente sind einfach nur langweilige Spiele. Und sei versichert, wenn du ein Spiel programmieren kannst, kannst du sicherlich auch ein Experiment programmieren.

1.4 Warum sollte ein Psychologe Programmieren lernen?

Warum sollte ein Psychologe, der an Menschen interessiert ist, lernen, wie man Computer programmiert? Die offensichtlichste Antwort ist, dass dies eine nützliche Fähigkeit ist. Wenn du programmieren kannst, hast du die Freiheit, ein Experiment zu erstellen, das deine Forschungsfrage beantwortet, und nicht ein Experiment, das aufgrund der Beschränkungen deiner Software implementiert werden kann.

Noch wichtiger, zumindest aus meiner Sicht, ist, dass das Erlernen des Programmierens die Art und Weise, wie du im Allgemeinen denkst, verändert. Menschen sind klug, aber Computer sind dumm[1]. Wenn du dein Experiment oder deine Reisepläne jemandem erklärst, kannst du ziemlich vage sein, einen kleinen Fehler machen, sogar bestimmte Teile überspringen. Menschen sind klug, daher werden sie die fehlenden Informationen mit ihrem Wissen ergänzen, einen Fehler finden und korrigieren, dich nach mehr Informationen fragen und selbst improvisieren können, sobald sie auf etwas stoßen, das du nicht abgedeckt hast. Computer sind dumm, also musst du präzise sein, du kannst keine Grauzonen haben, du kannst nichts dem "es wird herausfinden, wenn es passiert" (es wird es nicht) überlassen. Meine persönliche Erfahrung, die von Psychologen bestätigt wurde, die das Programmieren gelernt haben, ist, dass es dir bewusst macht, wie vage und ungenau Menschen sein können, ohne es zu merken (und ohne dass du es merkst). Programmieren zwingt dich dazu, präzise und gründlich zu sein, für alle Eventualitäten zu planen, die es geben könnte. Und das ist an sich eine sehr nützliche Fähigkeit, da sie auf alle Aktivitäten angewendet werden kann, die Planung erfordern, sei es ein experimentelles Design oder Reisevorkehrungen.

1.5 Warum Python?

Es gibt viele Möglichkeiten, ein Experiment für psychologische Forschung zu erstellen. Du kannst Drag-and-Drop-Systeme verwenden, entweder kommerzielle

[1] Dies wurde geschrieben, bevor große Sprachmodelle aufkamen, aber trifft immer noch zu, wenn es ums Programmieren geht.

wie Presentation, Experiment Builder oder kostenlose wie PsychoPy Builder Interface. Sie haben eine viel flachere Lernkurve, sodass du schneller mit dem Erstellen und Durchführen deiner Experimente beginnen kannst. Ihre einfache Handhabung hat jedoch einen Preis: Sie sind recht begrenzt, welche Reize du verwenden kannst und wie du die Präsentation, Bedingungen, Feedback etc. steuern kannst. In der Regel erlauben sie dir, sie zu erweitern, indem du das gewünschte Verhalten programmierst, aber du musst wissen, wie man das macht (Python-Kenntnisse bereichern deine PsychoPy-Experimente). Daher denke ich, dass diese Systeme, insbesondere PsychoPy, großartige Werkzeuge sind, um schnell ein einfaches Experiment zusammenzusetzen. Sie sind jedoch am nützlichsten, wenn du verstehst, *wie* sie den zugrunde liegenden Code erstellen und wie du ihn selbst programmieren würdest. Dann wirst du nicht durch die Software eingeschränkt, da du weißt, dass du etwas programmieren kannst, was der Standard-Drag-and-Drop nicht zulässt. Gleichzeitig kannst du dich immer dafür entscheiden, wenn Drag-and-Drop ausreichend, aber schneller ist, oder du kannst eine Kombination aus beiden Ansätzen verwenden. Am Ende geht es darum, Optionen und kreative Freiheit zu haben, ein Experiment zu programmieren, das deine Forschungsfrage beantwortet, und nicht nur ein Experiment, das deine Software erlaubt zu programmieren.

Wir werden das Programmieren in Python lernen, eine großartige Sprache, die eine einfache und klare Syntax mit der Kraft und Fähigkeit kombiniert, fast jedes Problem zu bewältigen. In diesem Seminar konzentrieren wir uns auf Desktop-Experimente, aber Sie können es für Online-Experimente (oTree und PsychoPy), wissenschaftliche Programmierung (NumPy und SciPy), Datenanalyse (pandas), Maschinelles Lernen (scikit-learn), Deep Learning (keras), Website-Programmierung (django), Computer Vision (OpenCV) usw. verwenden. Daher ist Python eines der vielseitigsten Programmierwerkzeuge, die Sie für alle Phasen Ihrer Forschung oder Arbeit verwenden können. Und Python ist kostenlos, so dass Sie sich keine Sorgen machen müssen, ob Sie oder Ihr zukünftiger Arbeitgeber die Lizenzgebühren bezahlen können (ein sehr reales Problem, wenn Sie Matlab verwenden).

1.6 Seminar-spezifische Informationen

Dies ist ein Material für das Seminar *Python für Sozial- und Experimentelle Psychologie*, wie ich es an der Universität Bamberg lehre. Jedes Kapitel behandelt ein einzelnes Spiel und führt die notwendigen Ideen ein und wird von Übungen begleitet, die Sie absolvieren und einreichen müssen. Um das Seminar zu bestehen, müssen Sie alle Aufgaben absolvieren, d.h., alle Spiele schreiben. Sie müssen nicht alle Übungen abschließen oder korrekte Lösungen für *alle* Übungen liefern, um den Kurs zu bestehen und Informationen darüber, wie die Punkte für Übungen in eine tatsächliche Note (falls Sie eine benötigen) oder "Bestanden" umgerechnet werden, werden während des Seminars verfügbar sein.

Das Material ist so strukturiert, dass jedes Kapitel oder Kapitelabschnitt typis-

cherweise einer einzelnen Sitzung entspricht, außer für die abschließenden Kapitel, die auf komplexeren Spielen basieren und daher mehr Zeit in Anspruch nehmen. Wir sind jedoch alle verschieden, also arbeiten Sie in Ihrem eigenen Tempo, lesen Sie das Material und reichen Sie Aufgaben eigenständig ein. Ich werde für jede Aufgabe detailliertes Feedback geben und Ihnen die Möglichkeit bieten, Probleme anzugehen und erneut einzureichen, ohne Punkte zu verlieren. Beachten Sie, dass mein Feedback nicht nur die tatsächlichen Probleme mit dem Code abdeckt, sondern auch die Art und Weise, wie Sie die Lösung implementiert haben und wie sauber und gut dokumentiert Ihr Code ist. Denken Sie daran, unsere Aufgabe ist es nicht nur, zu lernen, wie man ein funktionierendes Spiel programmiert, sondern wie man einen schönen, klaren, leicht zu lesenden und zu wartenden Code schreibt[2].

Sehr wichtig: Zögern Sie nicht, Fragen zu stellen. Wenn ich das Gefühl habe, dass Sie die Informationen im Material übersehen haben, werde ich Sie auf die genaue Stelle hinweisen. Wenn Sie verwirrt sind, werde ich Sie sanft mit Fragen dazu anregen, Ihr eigenes Problem zu lösen. Wenn Sie mehr Informationen benötigen, werde ich diese liefern. Wenn Sie einfach mehr wissen wollen, fragen Sie und ich werde erklären, warum die Dinge so sind, wie sie sind, oder vorschlagen, was Sie lesen sollten. Wenn ich das Gefühl habe, dass Sie das Problem ohne meine Hilfe lösen können, werde ich Ihnen das sagen (obwohl ich wahrscheinlich trotzdem noch ein paar andeutende Fragen stellen würde).

1.7 Über das Material

[2]Gute Gewohnheiten! Bilden Sie gute Gewohnheiten! Danke, dass Sie diese unterschwellige Botschaft gelesen haben.

Chapter 2

Software

Für dieses Buch und das Seminar müssen wir installieren

- PsychoPy.
- IDE Ihrer Wahl. Meine Anleitungen werden für Visual Studio Code sein, das eine sehr gute Python-Unterstützung bietet.
- Jupyter Notebook zum Ausprobieren kleiner Code-Snippets.

Ich werde keine detaillierten Anleitungen zur Installation der notwendigen Software geben, sondern Sie eher auf die offiziellen Handbücher verweisen. Dies macht diesen Text zukunftssicherer, da sich spezifische Details leicht ändern könnten[1].

2.1 PsychoPy

Wenn Sie Windows verwenden, laden Sie die Standalone PsychoPy Version herunter und installieren Sie diese. Verwenden Sie die neueste (und beste) Ihnen vorgeschlagene PsychoPy-Version (PsychoPy 2023.2.2 mit Python 3.8 zum Zeitpunkt des Schreibens) und folgen Sie den Anweisungen.

Wenn Sie Mac oder Linux verwenden, sind die Installation von PsychoPy über pip oder Anaconda Ihre Optionen. Bitte folgen Sie den aktuellen Anweisungen.

2.2 VS Code

Visual Studio Code ist ein kostenloser, leichtgewichtiger Open-Source-Editor mit starker Unterstützung für Python. Laden Sie den Installer für Ihre Plattform herunter und folgen Sie den Anweisungen.

[1] Wenn Sie Teil des Seminars sind, fragen Sie mich, wann immer Sie Probleme haben oder unsicher sind, wie Sie vorgehen sollen

Befolgen Sie als nächstes das Tutorial Getting Started with Python in VS Code. Wenn Sie Windows und die Standalone-Installation von PsychoPy verwenden, **überspringen** Sie den Abschnitt *Install a Python interpreter*, da Sie bereits eine Python-Installation haben, die mit PsychoPy gebündelt ist. Dies ist der Interpreter, den Sie im Abschnitt *Select a Python interpreter* verwenden sollten. In meinem Fall ist der Pfad `C:\Program Files\PsychoPy3\python.exe`.

Installieren und aktivieren Sie einen Linter, eine Software, die syntaxtische und stilistische Probleme in Ihrem Python-Quellcode hervorhebt. Folgen Sie dem Handbuch auf der Webseite von VS Code.

2.3 Jupyter Notebooks

Jupyter Notebooks bieten eine sehr bequeme Möglichkeit, Text, Bilder und Code in einem einzigen Dokument zu mischen. Sie erleichtern auch das Ausprobieren verschiedener kleiner Code-Snippets parallel ohne das Ausführen von Skripten. Wir werden uns für unser erstes Kapitel und gelegentliche Übungen oder Code-Tests später darauf verlassen. Es gibt zwei Möglichkeiten, wie Sie sie verwenden können: 1) in VS Code mit der Jupyter-Erweiterung, 2) in Ihrem Browser mit der klassischen Oberfläche.

2.3.1 Jupyter Notebooks in VS Code

Folgen Sie der Anleitung, wie Sie das Jupyter-Paket installieren und Notebooks in VS Code verwenden.

2.3.2 Jupyter Notebooks in Anaconda

Die einfachste Möglichkeit, Jupyter Notebooks zusammen mit vielen anderen nützlichen Data-Science-Tools zu verwenden, ist über das Anaconda Toolkit. Beachten Sie jedoch, dass dies eine *zweite* Python-Distribution in Ihrem System installiert. Dies könnte wiederum zu Verwirrung führen, wenn Sie mit Skripten in VS Code arbeiten und versehentlich den Anaconda-Interpreter statt den PsychoPy-Interpreter aktiv haben. Keine Panik, folgen Sie den Select a Python interpreter Anweisungen und stellen Sie sicher, dass Sie den PsychoPy-Interpreter als den aktiven haben.

Ansonsten laden Sie Anaconda herunter und installieren Sie es. Die Website hat einen ausgezeichneten Getting started Abschnitt.

2.4 Ordnung halten

Bevor wir anfangen, schlage ich vor, dass Sie einen Ordner namens *games-with-python* (oder so ähnlich) erstellen. Wenn Sie sich dafür entschieden haben, Jupyter Notebooks über Anaconda zu nutzen, sollten Sie diesen Ordner in Ihrem

Benutzerordner erstellen, da Anaconda dort die Dateien erwartet. Dann erstellen Sie einen neuen Unterordner für jedes Kapitel / Spiel. Für das Seminar müssten Sie einen Ordner mit allen Dateien zippen und hochladen.

Chapter 3

Programmier-Tipps und -Tricks

Bevor Sie Ihren ersten Code schreiben, müssen wir über die Kunst des Programmierens sprechen. Wie ich bereits erwähnt habe, geht es nicht nur darum, dass der Code funktioniert, sondern darum, dass er leicht zu verstehen ist. Korrekt funktionierender Code ist ein schönes Plus, aber wenn ich zwischen einem Spaghetti-Code, der momentan korrekt funktioniert, und einem klar geschriebenen und dokumentierten Code, der noch repariert werden muss, wählen muss, werde ich immer Letzteren bevorzugen. Ich kann Dinge reparieren, die ich verstehe, ich kann nur hoffen, wenn ich das nicht tue.

Unten sind einige Tipps zum Schreiben und Lesen von Code. Einige mögen beim ersten Lesen kryptisch klingen (sie werden klar, sobald wir das notwendige Material behandeln). Einige werden sich für die einfachen Projekte, die wir umsetzen werden, als Overkill anfühlen. Ich schlage vor, dass Sie diesen Abschnitt beim ersten Mal ungezwungen lesen, aber oft darauf zurückkommen, sobald wir ernsthaft mit Programmieren beginnen. Leider werden diese Tricks nicht funktionieren, wenn Sie sie nicht benutzen! Daher sollten Sie sie *immer* benutzen und sie sollten Ihre *guten Gewohnheiten* werden, wie das Anlegen eines Sicherheitsgurtes. Der Sicherheitsgurt ist an den meisten (hoffentlich allen) Tagen nicht nützlich, aber Sie tragen ihn, weil er plötzlich und sehr dringend extrem nützlich werden könnte und Sie nie sicher sein können, wann das passieren wird. Gleiches gilt für das Programmieren. Oft werden Sie versucht sein, einen "quick-n-dirty" Code zu schreiben, weil es sich nur um einen "einfachen Test", eine temporäre Lösung, einen Prototyp, ein Pilotexperiment, usw. handelt. Aber wie man so schön sagt: "Es gibt nichts Beständigeres als eine provisorische Lösung". Häufiger als nicht werden Sie feststellen, dass Ihr Spielzeugcode zu einem ausgewachsenen Experiment herangewachsen ist und es ein Durcheinander ist. Oder Sie möchten zu dem Pilotexperiment zurückkehren, das Sie vor

ein paar Monaten durchgeführt haben, aber Sie stellen fest, dass es einfacher ist, von vorne zu beginnen, als zu verstehen, wie dieses Monster funktioniert[1]. Widerstehen Sie also der Versuchung! Bilden Sie gute Gewohnheiten und Ihr zukünftiges Ich wird sehr dankbar sein!

3.1 Den Code schreiben

3.1.1 Verwende einen Linter

Ein Linter ist ein Programm, das deinen Code-Stil analysiert und highlightet Probleme, die es findet: unnötige Leerzeichen, fehlende Leerzeichen, falsche Namen, übermäßig lange Zeilen, usw. Diese haben keinen Einflluss darauf, wie der Code ausgeführt wird, aber das Befolgen der Linter-Ratschläge führt zu einem konsistenten Standard, wenn auch langweiligen[^ "Langweilig ist gut!", siehe den Film "The Hitman's Bodyguard."] Python Code. Versuche, alle Probleme zu beheben, die der Linter aufgeworfen hat. Verwende jedoch dein besseres Urteilsvermögen, denn manchmal sind Zeilen, die länger sind als der Linter es bevorzugen würde, lesbarer als zwei kürzere. Ebenso kann ein "schlechter" Variablenname nach den Standards des Linters für einen Psychologen eine aussagekräftige Bezeichnung sein. Denke daran, dein Code ist für Menschen, nicht für den Linter.

3.1.2 Dokumentiere deinen Code

Jedes Mal, wenn du eine neue Datei erstellst: dokumentiere sie und aktualisiere die Dokumentation, wann immer du neue Funktionen oder Klassen hinzufügst/änderst/löschst. Jedes Mal, wenn du eine neue Funktion erstellst: dokumentiere sie. Neue Klasse: dokumentiere sie. Neue Konstante: es sei denn, es ist alleine aus dem Namen klar, dokumentiere sie. Du wirst eine NumPy Methode der Dokumentation in dem Buch lernen.

Ich kann nicht genug betonen, wie wichtig es ist, deinen Code zu dokumentieren. VS Code (ein Editor, den wir verwenden werden) ist intelligent genug, um NumPy Docstrings zu parsen, also wird es dir diese Hilfe anzeigen, wann immer du deine eigenen Funktionen verwendest (hilft dir, dir selbst zu helfen!). Noch wichtiger ist, dass das Schreiben von Dokumentationen dich dazu zwingt, in menschlicher Sprache zu denken und zu formulieren, was die Funktion oder Klasse macht, welchen Typ die Argumente / Attribute / Methoden haben, was der Bereich der gültigen Werte ist, was die Standards sind, was eine Funktion zurückgeben sollte usw. Mehr als oft nicht, wirst du feststellen, dass du ein wichtiges Detail übersehen hast, das aus dem Code selbst nicht ersichtlich ist.

[1] Mir ist das öfter passiert, als ich zugeben möchte.

3.1.3 Füge etwas Luft hinzu

Trenne Codeblöcke durch einige leere Zeilen. Denke an Absätze im normalen Text. Du würdest nicht wollen, dass dein Buch ein einziger Absatz-Alptraum ist? Platziere vor jedem Codeblock einen Kommentar, der erklärt, *was* er macht, aber nicht *wie* er es macht. Zum Beispiel wird es in unserem typischen PsychoPy-basierten Spiel einen Punkt geben, an dem wir alle Reize zeichnen und das Fenster aktualisieren. Das ist ein schöner selbstständiger Codeblock, der als `# Zeichnen aller Reize` beschrieben werden kann. Der Code liefert Details darüber, was genau gezeichnet wird, was die Zeichenreihenfolge ist, usw. Aber dieser einzelne Kommentar hilft dir zu verstehen, worum es in diesem Codeblock geht und ob er für dich momentan relevant ist. Das Gleiche gilt für `# Verarbeitung von Tastendrücken` oder `# Überprüfung der Spielende-Bedingungen`, usw. Aber sei vorsichtig und stelle sicher, dass der Kommentar den Code korrekt beschreibt. Wenn der Kommentar zum Beispiel `# Zeichnen aller Reize` sagt, sollte es nirgendwo anders einen Code zum Zeichnen von Reizen geben und keinen weiteren Code, der etwas anderes tut!

3.1.4 Schreibe deinen Code Schritt für Schritt

Dein Motto sollte "langsam aber stetig" lauten. Auf diese Weise werde ich dich durch die Spiele führen. Beginne immer mit etwas extrem Einfachem wie einem statischen Rechteck oder Bild. Stelle sicher, dass es funktioniert. Füge eine kleinere Funktionalität hinzu: Farbwechsel, Positionsänderung, ein weiteres Rechteck, speichere es als Attribut, usw. Stelle sicher, dass es funktioniert. Gehe niemals zum nächsten Schritt, es sei denn, du verstehst vollständig, was dein aktueller Code macht und du bist dir zu 100% sicher, dass er sich so verhält, wie er sollte. Und ich meine 100% ernst! Wenn du auch nur den Schatten eines Zweifels hast, überprüfe es erneut. Andernfalls wird dieser Schatten wachsen und dich zunehmend unsicher über deinen Code machen. Dieser Ansatz in Schildkrötengeschwindigkeit mag albern und übermäßig langsam erscheinen, ist aber immer noch schneller, als einen großen Codeblock zu schreiben und dann zu versuchen, ihn zum Laufen zu bringen. Es ist viel einfacher, einfache Probleme einzeln zu lösen, als viele auf einmal.

3.1.5 Es ist nichts Falsches an StackOverflow

Ja, man kann immer versuchen, eine Lösung für sein Problem auf StackOverflow zu finden[2]. Ich mache das die ganze Zeit! Sie sollten die bereitgestellte Lösung jedoch *nur verwenden, wenn Sie sie verstehen*! Kopieren Sie nicht einfach den Code, der ein Problem wie Ihres zu lösen *scheint*. Wenn Sie das tun und Sie haben Glück, könnte es funktionieren. Oder, wieder wenn Sie Glück haben, funktioniert es auf eine offensichtliche Weise nicht. Aber wenn Sie nicht so viel Glück haben, wird es (manchmal) subtil falsch funktionieren. Und da Sie nicht wirklich wussten, was der Code tat, als Sie ihn einfügten, werden Sie noch mehr

[2]Wenn Sie jedoch an dem Seminar teilnehmen, fragen Sie mich bitte zuerst!

verwirrt sein. Nutzen Sie also StackOverflow als eine Quelle des Wissens, nicht als eine Quelle zum Kopieren und Einfügen von Code!

3.2 Den Code lesen

Das Lesen von Code ist einfach, weil Computer dumm sind und Sie schlau sind. Das bedeutet, dass die Anweisungen, die Sie dem Computer geben, notwendigerweise sehr einfach sind und daher für einen Menschen sehr leicht zu verstehen sind. Das Lesen von Code ist jedoch auch schwierig, weil Computer dumm sind und Sie schlau sind. Sie sind so schlau, dass Sie nicht einmal den gesamten Code lesen müssen, um zu verstehen, was er tut. Sie lesen nur die Schlüsselstellen und füllen die Lücken aus. Leider neigen Sie dazu, Fehler zu überlesen. Dies ist nicht nur beim Programmieren der Fall, wenn Sie jemals einen Text Korrektur gelesen haben, wissen Sie, wie schwierig es ist, Tippfehler zu finden. Ihr Gehirn korrigiert sie in Echtzeit mit Hilfe des Kontexts und Sie lesen das Wort so, wie es sein sollte, nicht so, wie es tatsächlich geschrieben ist[3].

Meine Erfahrung mit Programmierung im Allgemeinen und im Besonderen auf diesem Seminar ist, dass die meisten Probleme, bei denen Sie stecken bleiben, im Nachhinein einfach bis dumm und offensichtlich sind[4]. Verzweifeln Sie nicht! Es liegt nicht an Ihnen, sondern ist lediglich eine Folge davon, wie wunderbar Ihr Gehirn für die Mustererkennung verdrahtet ist. Im Folgenden finden Sie einige Vorschläge, die Ihnen helfen könnten, das Lesen von Code robuster zu gestalten.

3.2.1 Denken Sie wie ein Computer

Lesen Sie den Code Zeile für Zeile und "führen Sie ihn aus", so wie es der Computer tun würde. Verwenden Sie Stift und Papier, um den Überblick über die Variablen zu behalten. Verfolgen Sie, welche Codeblöcke wann erreicht werden können. Verlangsamen Sie sich und stellen Sie sicher, dass Sie jede Zeile verstehen und in der Lage sind, den Überblick über die Variablen zu behalten. Sobald Sie das tun, wird es einfach sein, einen Fehler zu erkennen.

3.2.2 Tun Sie so, als hätten Sie diesen Code in Ihrem Leben noch nie gesehen

Nehmen Sie an, Sie haben keine Ahnung, was der Code tut. Wie ich schrieb, sehen Sie oft *wörtlich* einen Fehler nicht, weil Ihr Gehirn Details auffüllt und die Realität so biegt, dass sie Ihren Erwartungen entspricht[5]. Sie *wissen*, was dieser

[3] Tipp: Lesen Sie Ihren Text ein Satz nach dem anderen von hinten nach vorne oder lesen Sie zufällig einen Satz nach dem anderen. Dies bricht den Fluss des Textes und hilft Ihnen, sich auf die Wörter zu konzentrieren, anstatt auf die Bedeutung und die Geschichte.

[4] Im Nachhinein ist man immer schlauer!

[5] Kürzlich verbrachte ich eine halbe Stunde damit, zu verstehen, warum zwei identische Codeblöcke unterschiedliche Ergebnisse liefern. Mein Sohn fand fast sofort einen Unterschied

Codeblock tun sollte, also lesen Sie ihn statt dessen nur flüchtig und nehmen an, dass er tut, was er soll, es sei denn, es sieht offensichtlich schrecklich falsch aus. Es ist schwer, Ihre Erwartungen auszuschalten, aber es ist immens hilfreich.

3.2.3 Suchen Sie nicht nur unter der Straßenlampe

Immer wenn Sie neuen Code verwenden oder etwas implementieren müssen, das kompliziert erscheint, und Ihr Code nicht so funktioniert, wie er sollte, neigen Sie dazu, anzunehmen, dass das Problem beim neuen, ausgefallenen Code liegt. Einfach, weil er neu, ausgefallen und kompliziert ist. Meiner Erfahrung nach versteckt sich der Fehler jedoch typischerweise in der einfacheren "trivialen" Codezeile in der Nähe, die Sie nie richtig anschauen, weil sie einfach und trivial ist. Überprüfen Sie alles, nicht nur die Stellen, an denen Sie einen Fehler vermuten.

3.2.4 Nutzen Sie den Debugger

Im Buch werden Sie lernen, wie Sie die Ausführung Ihres Spiels anhalten können, um seinen Zustand zu untersuchen. Nutzen Sie dieses Wissen! Setzen Sie Haltepunkte und führen Sie den Code Schritt für Schritt aus. Überprüfen Sie die Werte von Variablen im Tab "Watch". Verwenden Sie die Debug-Konsole, um zu überprüfen, ob Funktionen die Ergebnisse liefern, die sie sollten. Teilen Sie komplexe Bedingungen oder mathematische Formeln in kleine Teile auf, kopieren und führen Sie diese Teile in der Debug-Konsole aus und überprüfen Sie, ob die Zahlen zusammenpassen. Stellen Sie sicher, dass ein Codeblock in Ordnung ist und analysieren Sie dann den nächsten. Das Debuggen ist besonders hilfreich, um den Code zu identifizieren, der nicht erreicht wird oder zum falschen Zeitpunkt erreicht wird.

3.3 Zen von Python

Ich fand den Zen von Python als gute Inspiration, um die Programmierung anzugehen.

(ein fehlendes Komma in einem von ihnen), weil es für ihn nur ein Haufen Buchstaben und Zahlen war.

Chapter 4

Python Grundlagen

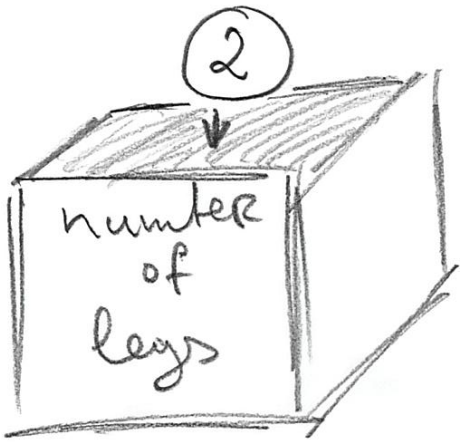
Hoffentlich hast du bereits einen speziellen Ordner für dieses Buch erstellt. Lade das Übungs-Notebook herunter (Alt+Klick sollte es eher herunterladen als öffnen), lege es in den Ordner des Kapitels und öffne es, siehe die relevanten Anweisungen. Du musst zwischen den Erklärungen hier und den Übungen im Notizbuch hin und her wechseln, also halte sie beide offen.

4.1 Konzepte des Kapitels

- Variablen.
- Konstanten.
- Grundlegende Werttypen.
- Dinge ausdrucken.
- Werte in Zeichenketten einfügen.

4.2 Variablen

Das erste grundlegende Konzept, mit dem wir uns vertraut machen müssen, ist die **Variable**. Variablen werden verwendet, um Informationen zu speichern, und du kannst sie dir als eine Kiste mit einem Namensschild vorstellen, in die du etwas hineinlegen kannst. Das Namensschild auf dieser Kiste ist der Name der Variable und ihr Wert ist das, was du darin speicherst. Zum Beispiel können wir eine Variable erstellen, die die Anzahl der Beine speichert, die ein Spielfigur hat. Wir beginnen mit einer für einen Menschen typischen Zahl.



In Python würdest du schreiben

```
anzahl_der_beine = 2
```

Die obige **Zuweisungsanweisung** hat eine sehr einfache Struktur:

```
<variablen-name> = <wert>
```

Der Variablenname (das Namensschild auf der Kiste) sollte aussagekräftig sein, er kann mit Buchstaben oder _ beginnen und kann Buchstaben, Zahlen und das Symbol _ enthalten, aber keine Leerzeichen, Tabs, Sonderzeichen usw. Python empfiehlt[1] dass du die **snake_case** Schreibweise (alles in Kleinbuchstaben, Unterstrich für Leerzeichen) verwendest, um deine Variablennamen zu formatieren. Der `<Wert>` auf der rechten Seite ist eine komplexere Geschichte, da er fest codiert sein kann (wie im obigen Beispiel), berechnet werden kann, indem andere Variablen oder dieselbe Variable, zurückgegeben von einer Funktion, usw. verwendet werden.

Die Verwendung von Variablen bedeutet, dass du dich auf die **Bedeutung** der entsprechenden Werte konzentrieren kannst, anstatt dir Sorgen darüber zu machen, was diese Werte sind. Beispielsweise kannst du das nächste Mal, wenn du etwas auf Grundlage der Anzahl der Beine eines Charakters berechnen musst (z.B., wie viele Paar Schuhe benötigt ein Charakter), dies auf Grundlage des aktuellen Wertes der Variablen `anzahl_der_beine` berechnen, anstatt anzunehmen, dass es `1` ist.

```
# SCHLECHT: Warum 1? Ist es, weil der Character zwei Beine hat oder
# weil wir jedem Character ein Paar Schuhe geben, unabhängig von
# seiner tatsächlichen Anzahl von Beinen?
paar_schuhe = 1

# BESSER (aber was, wenn unser Charakter nur ein Bein hat?)
```

[1] Naja, eigentlich besteht es darauf.

```
paar_schuhe = anzahl_der_beine / 2
```

Variablen geben dir auch Flexibilität. Ihre Werte können sich während des Programmablaufs ändern: Der Punktestand des Spielers steigt, die Anzahl der Leben nimmt ab, die Anzahl der Zauber, die er wirken kann, steigt oder fällt je nach ihrem Einsatz, usw. Dennoch kannst du immer den Wert in der Variablen verwenden, um die notwendigen Berechnungen durchzuführen. Hier ist zum Beispiel ein leicht erweitertes Beispiel für `anzahl_der_schuhe`.

```
anzahl_der_beine = 2

# ...
# etwas passiert und unser Charakter wird in einen Tintenfisch verwandelt
anzahl_der_beine = 8
# ...

# der gleiche Code funktioniert immer noch und wir können immer noch die richtige Anzahl an Schuh
paar_schuhe = anzahl_der_beine / 2
```

Wie bereits erwähnt, kannst du eine Variable als eine beschriftete Kiste betrachten, in die du etwas hineinlegen kannst. Das bedeutet, dass du immer den alten Wert "wegwerfen" und etwas Neues hineinlegen kannst. Im Falle von Variablen geschieht der "Wegwerf"-Teil automatisch, da ein neuer Wert den alten überschreibt. Überprüfe selbst, welcher der endgültige Wert der Variable im unten stehenden Code ist?

```
anzahl_der_beine = 2
anzahl_der_beine = 5
anzahl_der_beine = 1
anzahl_der_beine
```

Mache Übung #1.

Beachte, dass eine Variable (eine "Kiste mit Namensschild") erst existiert, nachdem du ihr etwas zugewiesen hast. Der folgende Code erzeugt also einen `NameError`, die Python-art zu sagen, dass sie noch nie von der Variable `anzahl_der_haende` gehört hat.

```
anzahl_der_beine = 2
anzahl_der_handschuhe = anzahl_der_haende / 2
```

Du kannst jedoch eine Variable erstellen, die keinen *spezifischen* Wert hat, indem du ihr `None` zuweist. `None` wurde speziell für die Bedeutung *kein Wert* oder *nichts* zur Sprache hinzugefügt.

```
anzahl_der_haende = None # Die Variable existiert jetzt, hat aber keinen speziellen Wert.
```

Wie du bereits gesehen hast, kannst du einen Wert *berechnen*, anstatt ihn anzugeben. Was wäre die Antwort hier?

```
anzahl_der_beine = 2 * 2
anzahl_der_beine = 7 - 2
anzahl_der_beine
```

Mache Übung #2.

4.3 Zuweisungen sind keine Gleichungen!

Sehr wichtig: obwohl Zuweisungen *wie* mathematische Gleichungen *aussehen*, sind sie **keine Gleichungen!** Sie folgen einer **sehr wichtigen** Regel, die Sie im Kopf behalten müssen, wenn Sie Zuweisungen verstehen: die rechte Seite eines Ausdrucks wird *zuerst* ausgewertet, bis der Endwert berechnet ist. Erst dann wird dieser Endwert der auf der linken Seite angegebenen Variable zugewiesen (also in die Kiste gelegt). Das bedeutet, dass Sie die gleiche Variable auf *beiden* Seiten verwenden können! Sehen wir uns diesen Code an:

```
x = 2
y = 5
x = x + y - 4
```

Was passiert, wenn der Computer die letzte Zeile auswertet? Zunächst nimmt er die *aktuellen* Werte aller Variablen (`2` für `x` und `5` für `y`) und setzt sie in den Ausdruck. Nach diesem internen Schritt sieht der Ausdruck so aus:

```
x = 2 + 5 - 4
```

Dann berechnet er den Ausdruck auf der rechten Seite und speichert, **sobald die Berechnung abgeschlossen ist**, diesen neuen Wert in `x`

```
x = 3
```

Machen Sie die Übung #3, um sicherzugehen, dass Sie dies verstanden haben.

4.4 Konstanten

Obwohl die eigentliche Stärke von Variablen darin besteht, dass Sie ihren Wert ändern können, sollten Sie sie auch dann verwenden, wenn der Wert im gesamten Programm konstant bleibt. In Python gibt es keine echten Konstanten, sondern die Übereinkunft, dass ihre Namen vollständig `GROSSGESCHRIEBEN` sein sollten. Entsprechend wissen Sie, wenn Sie `SOLCH_EINE_VARIABLE` sehen, dass Sie ihren Wert nicht ändern sollten. Technisch gesehen ist das nur eine Empfehlung, denn niemand kann Sie davon abhalten, den Wert einer `KONSTANTE` zu ändern. Aber ein großer Teil der Benutzerfreundlichkeit von Python resultiert aus solchen Übereinkünften (wie der `snake_case` Konvention oben). Wir werden später mehr von solchen Übereinkünften treffen, zum Beispiel beim Lernen über Objekte.

Unter Berücksichtigung all dessen, wenn die Anzahl der Beine im Spiel konstant bleibt, sollten Sie diese Konstanz betonen und schreiben

```
ANZAHL_DER_BEINE = 2
```

Ich empfehle dringend die Verwendung von Konstanten und vermeide das Hardcoding von Werten. Erstens, wenn Sie mehrere identische Werte haben, die verschiedene Dinge bedeuten (2 Beine, 2 Augen, 2 Ohren, 2 Fahrzeuge pro Figur, etc.), wird Ihnen eine `2` im Code nicht verraten, was diese `2` bedeutet (die Beine? Die Ohren? Der Punktemultiplikator?). Sie können das natürlich herausfinden, basierend auf dem Code, der diese Nummer verwendet, aber Sie könnten sich diese zusätzliche Mühe ersparen und stattdessen eine ordnungsgemäß benannte Konstante verwenden. Dann lesen Sie einfach ihren Namen und die Bedeutung des Wertes wird offensichtlich, und es ist die Bedeutung und nicht der tatsächliche Wert, der Sie hauptsächlich interessiert. Zweitens, wenn Sie entscheiden, diesen Wert dauerhaft zu *ändern* (sagen wir, unsere Hauptfigur ist jetzt ein Dreifuß), bedeutet die Verwendung einer Konstante, dass Sie sich nur an einer Stelle Sorgen machen müssen, der Rest des Codes bleibt unverändert. Wenn Sie diese Zahl hart codiert haben, erwartet Sie eine aufregende [2] und definitiv lange Suche und Ersetzung im gesamten Code.

Machen Sie die Übung #4.

4.5 Datentypen

Bisher haben wir nur ganzzahlige numerische Werte verwendet (1, 2, 5, 1000…). Obwohl Python viele verschiedene Datentypen unterstützt, konzentrieren wir uns zunächst auf eine kleine Auswahl davon:

- Ganze Zahlen, die wir bereits verwendet haben, z.B. `-1`, `100000`, `42`.
- Fließkommazahlen, die jeden realen Wert annehmen können, z.B. `42.0`, `3.14159265359`, `2.71828`.
- Zeichenketten, die Text speichern können. Der Text ist zwischen entweder gepaarten Anführungszeichen `"einiger Text"` oder Apostrophen `'einiger Text'` eingeschlossen. Das bedeutet, dass Sie Anführungszeichen oder Apostrophe innerhalb der Zeichenkette verwenden können, solange sie von der Alternative umschlossen ist. Z.B., `"Schüleraufgaben"` (eingeschlossen in `"`, Apostroph `'` innen) oder `'"Alle Verallgemeinerungen sind falsch, auch diese." Mark Twain'` (Zitat von Apostrophen eingeschlossen). Es gibt noch viel mehr zu Zeichenketten und wir werden dieses Material im Laufe des Kurses behandeln.
- Logische / boolesche Werte, die entweder `True` oder `False` sind.

Bei der Verwendung einer Variable ist es wichtig, dass Sie wissen, welchen Datentyp sie speichert, und das liegt meist bei Ihnen. In einigen Fällen wird Python

[2] nicht wirklich

einen Fehler ausgeben, wenn Sie versuchen, eine Rechnung mit inkompatiblen Datentypen durchzuführen. In anderen Fällen wird Python Werte automatisch zwischen bestimmten Typen konvertieren, z.B. ist jeder Ganzzahlwert auch ein Realwert, so dass die Konvertierung von `1` zu `1.0` meist trivial und automatisch ist. In anderen Fällen müssen Sie jedoch möglicherweise eine explizite Konvertierung verwenden. Gehen Sie zur Übung #5 und versuchen Sie zu erraten, welcher Code laufen wird und welcher einen Fehler wegen inkompatiblen Typen werfen wird?

```
5 + 2.0
'5' + 2
'5' + '2'
'5' + True
5 + True
```

Mache Übung #5.

Überrascht vom letzten? Das liegt daran, dass intern `True` auch `1` und `False` `0` ist!

Sie können explizit von einem Typ in einen anderen umwandeln, indem Sie spezielle Funktionen verwenden. Beispielsweise können Sie eine Zahl oder einen logischen Wert in einen String umwandeln, indem Sie einfach str(<value>) schreiben. Was wäre das Ergebnis in den untenstehenden Beispielen?

```
str(10 / 2)
str(2.5 + True)
str(True)
```

Mache Übung #6.

Ähnlich können Sie mit der Funktion bool(<value>) in eine logische/boolesche Variable umwandeln. Die Regeln sind einfach, für numerische Werte ist `0` gleich `False`, jeder andere Nicht-Null-Wert wird in `True` umgewandelt. Für Zeichenketten wird eine leere Zeichenkette `''` als `False` bewertet und eine nicht leere Zeichenkette wird in `True` umgewandelt. Was wäre die Ausgabe in den untenstehenden Beispielen?

```
bool(-10)
bool(0.0)

secret_message = ''
bool(secret_message)

bool('False')
```

Mache Übung #7.

Die Umwandlung in Ganzzahlen oder Fließkommazahlen mit int(<value>) bzw. float(<value>) ist komplizierter. Der einfachste Fall ist von logisch auf Ganz-

zahl/Fließkommazahl, da `True` Ihnen `int(True)` ist `1` und `float(True)` ist `1.0` gibt und `False` gibt Ihnen `0`/`0.0`. Beim Umwandeln von Fließkommazahl auf Ganzzahl lässt Python einfach den Bruchteilteil fallen (es rundet nicht richtig!). Bei der Umwandlung einer Zeichenkette muss es sich um eine gültige Zahl des entsprechenden Typs handeln, sonst wird ein Fehler erzeugt. Sie können z. B. eine Zeichenkette wie `"123"` in eine Ganzzahl oder eine Fließkommazahl umwandeln, aber das funktioniert nicht für `"a123"`. Darüber hinaus können Sie `"123.4"` in eine Fließkommazahl umwandeln, aber nicht in eine Ganzzahl, da sie einen Bruchteil enthält. Angesichts all dessen, welche Zellen würden funktionieren und welche Ausgabe würden sie erzeugen?

```
float(False)
int(-3.3)
float("67.8")
int("123+3")
```

Mache Übung #8.

4.6 Ausgabe drucken

Um den Wert auszudrucken, müssen Sie die Funktion print() verwenden (wir werden später allgemein über Funktionen sprechen). Im einfachsten Fall übergeben Sie den Wert und er wird ausgegeben.

```
print(5)
#> 5
```

oder

```
print("fünf")
#> fünf
```

Natürlich wissen Sie bereits über die Variablen Bescheid, also statt den Wert direkt einzugeben, können Sie stattdessen eine Variable übergeben und ihr *Wert* wird ausgegeben.

```
anzahl_der_pfannkuchen = 10
print(anzahl_der_pfannkuchen)
#> 10
```

oder

```
frühstück = "pfannkuchen"
print(frühstück)
#> pfannkuchen
```

Sie können auch mehr als einen Wert/Variablen an die Druckfunktion übergeben und alle Werte werden nacheinander gedruckt. Wenn wir dem Benutzer zum Beispiel sagen wollen, was ich zum Frühstück hatte, können wir das tun

```
frühstück = "pfannkuchen"
anzahl_der_artikel = 10
print(frühstück, anzahl_der_artikel)
#> pfannkuchen 10
```

Was wird von dem untenstehenden Code gedruckt?

```
abendessen = "steak"
zähler = 4
nachtisch = "muffins"

print(zähler, abendessen, zähler, nachtisch)
```

Mache Übung #9.

Allerdings möchten Sie wahrscheinlich expliziter sein, wenn Sie die Informationen ausdrucken. Stellen Sie sich zum Beispiel vor, Sie haben diese drei Variablen:

```
mahlzeit = "Frühstück"
gericht = "Pfannkuchen"
anzahl = 10
```

Sie könnten natürlich `print(mahlzeit, gericht, anzahl)` machen, aber es wäre schöner, "*Ich hatte* ***10*** ***Pfannkuchen*** *zum* ***Frühstück***" zu drucken, wobei die in Fettschrift gedruckten Elemente die eingefügten Variablenwerte wären. Dafür müssen wir die Formatierung von Zeichenketten verwenden. Bitte beachten Sie, dass die Formatierung von Zeichenketten nicht spezifisch für das Drucken ist, Sie können einen neuen Zeichenkettenwert über die Formatierung erstellen und ihn in einer Variable speichern, ohne ihn auszudrucken, oder ihn ausdrucken, ohne ihn zu speichern.

4.7 Formatierung von Zeichenketten

Eine großartige Ressource zur Formatierung von Zeichenketten in Python ist pyformat.info. Da sich Python ständig weiterentwickelt, gibt es nun mehr als eine Art, Zeichenketten zu formatieren. Im Folgenden werde ich das "alte" Format vorstellen, das auf der klassischen Formatierung von Zeichenketten basiert, die in der Funktion `sprintf` in C, Matlab, R und vielen anderen Programmiersprachen verwendet wird. Es ist etwas weniger flexibel als neuere, aber für einfache Aufgaben ist der Unterschied vernachlässigbar. Das Wissen über das alte Format ist nützlich wegen seiner Allgemeinheit. Wenn Sie Alternativen lernen möchten, lesen Sie unter dem oben angegebenen Link.

Der allgemeine Aufruf lautet `"ein String mit Formatierung"%(Tupel von Werten, die während der Formatierung verwendet werden)`. Sie werden später mehr über Tupel lernen. Gehen Sie im Moment davon aus, dass es sich einfach um eine durch Kommas getrennte Liste von Werten handelt, die in runden Klammern eingeschlossen sind: `(1, 2, 3)`.

In `"ein String mit Formatierung"`, geben Sie an, wo Sie den Wert mit dem Zeichen `%` einfügen möchten, das von einer *optionalen* Formatierungsinformation und dem *erforderlichen* Symbol, das den **Typ** des Wertes definiert, gefolgt wird. Die Typsymbole sind

- `s` für Zeichenkette
- `d` für eine Ganzzahl
- `f` für einen Fließkommawert
- `g` für einen "optimal" gedruckten Fließkommawert, so dass für große Werte die wissenschaftliche Notation verwendet wird (*z.B.*, `10e5` statt `100000`).

Hier ist ein Beispiel, wie man einen String mit einer Ganzzahl formatiert:

```
print("Ich hatte %d Pfannkuchen zum Frühstück"%(10))
#> Ich hatte 10 Pfannkuchen zum Frühstück
```

Sie sind nicht darauf beschränkt, einen einzigen Wert in einen String einzufügen. Sie können weitere Positionen über `%` angeben, müssen jedoch sicherstellen, dass Sie die richtige Anzahl von Werten in der richtigen Reihenfolge übergeben. Können Sie vor dem Ausführen herausfinden, welcher Aufruf tatsächlich funktioniert (und was die Ausgabe sein wird) und welcher einen Fehler verursacht?

```
print('Ich hatte %d Pfannkuchen und entweder %d oder %d Steaks zum Abendessen'%(2))
print('Ich hatte %d Pfannkuchen und %d Steaks zum Abendessen'%(7, 10))
print('Ich hatte %d Pfannkuchen und %d Steaks zum Abendessen'%(1, 7, 10))
```

Machen Sie Übung #10.

Wie oben erwähnt, haben Sie im Falle von echten Werten zwei Möglichkeiten: `%f` und `%g`. Letzterer verwendet die wissenschaftliche Notation (z.B. `1e10` für `10000000000`), um eine Darstellung kompakter zu machen.

Machen Sie Übung #11, um ein besseres Gefühl für den Unterschied zu bekommen.

Es gibt noch viel mehr zur Formatierung und Sie können auf pyformat.info darüber lesen. Diese Grundlagen sind jedoch ausreichend, um in dem nächsten Kapitel mit der Programmierung unseres ersten Spiels zu beginnen.