Kurzfassung: “Python und Zope als Unterrichtswerkzeuge“

Geht es um das Aneignen von Programmierkenntnissen, ist die momentane Situation nach Ansicht des Autors nicht optimal. Sie ist verbesserungswürdig. Gerade in der schnelllebigen Informatik sollte im Unterricht nicht auf Trends oder Modeerscheinungen zurückgegriffen werden, sondern die fundamentalen Ideen der Wissenschaft Informatik mit einem guten Konzept an die Studierenden herangetragen werden. Erste Programmierkenntnisse mit z.B. C oder Java zu vermitteln bzw. vermittelt zu bekommen, ist problematisch. Diese Arbeit hat zum Ziel, einen einfacheren Weg aufzeigen und soll begründen, warum erste Programmiererfahrungen mit Python didaktisch sinnvoller sind.

1 Fundamentale Ideen als Unterrichtsprinzip

Eine Argumentationsgrundlage dieser Arbeit ist die Definition einer *fundamentalen Idee* nach Schwill [Sch93]. In seiner Arbeit untersucht er die philosophische Sicht der *Idee* nach Plato und Kant, und formuliert diese mit den *Strukturen* nach J.S. Bruner [Bru60] zur fundamentalen Idee. J.S. Bruner hat das didaktische Prinzip an den Strukturen der zugrundeliegenden Wissenschaft, an denen sich der Unterricht orientieren soll, formuliert.

> *“Eine **fundamentale Idee** (bezgl. einer Wissenschaft) ist ein Denk-, Handlungs-, Beschreibungs- oder Erklärungsschema, das*
>
> 1. *in verschiedenen Bereichen (der Wissenschaft) vielfältig anwendbar oder erkennbar ist (Horizontalkriterium),*
> 2. *auf jedem intellektuellen Niveau aufgezeigt und vermittelt werden kann (Vertikalkriterium),*
> 3. *in der historischen Entwicklung (der Wissenschaft) deutlich wahrnehmbar ist und längerfristig relevant bleibt (Zeitkriterium),*
> 4. *einen Bezug zu Sprache und Denken des Alltags und der Lebenswelt besitzt (Sinnkriterium).“*

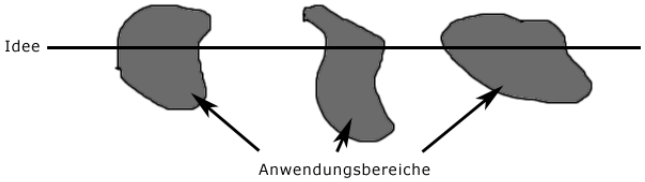

Abbildung 1: Horizontalkriterium nach [Sch93]

Das *Horizontalkriterium* veranschaulicht Schwill wie in Abbildung 1. Eine Idee wird soweit abstrahiert, dass fachspezifische Elemente herausfallen. Damit kann themen- und fachübergreifend gearbeitet werden. Der Lernende erkennt durch immer wiederkehrende Prinzipien die übergreifende Relevanz der Thematik. Anschließend wird durch Wiederholung das Wichtige gefestigt. Die gemeinsame Idee kann jedoch wiederum nur durch umfassendes Spezialwissen herausgearbeitet werden. Das ist die Aufgabe der Lehrkräfte.

Das *Vertikalkriterium* kann wie ein Faden im Bildungsweg gesehen werden. Diesem Faden wird im Laufe der Ausbildung gefolgt. Dabei steigen das Niveau und die Detaillierung der Materie (Abbildung 2).

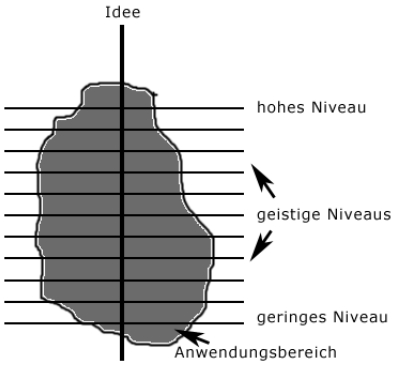

Abbildung 2: Vertikalkriterium nach [Sch93]

Das *Zeitkriterium* ist in der Informatik nicht schwer zu erfüllen. Die Schnelllebigkeit der Technik birgt zwar viele Modeerscheinungen, jedoch sichert das Kriterium die Kontinuität des Unterrichts und den Wert der Kenntnisse und Erfahrungen der Unterrichtenden und verhindert fachliche Moden.[Mod02]

Dabei ist die Trägheit der Lehre eine Art Vorauslese, um das Zeitkriterium einer fundamentalen Idee zu erfüllen.

Der Autor interpretiert das *Sinnkriterium* als das tatsächliche Wertempfinden beim Empfänger der Idee, beim Lernenden. Wie gut kann die Thematik als praxisrelevant oder als sinnvoll erkannt werden? Kann der Bezug zur späteren Verwendung in der Arbeitswelt hergestellt werden?

Der Einsatz fundamentaler Ideen wird von [Mod02] wie folgt beschrieben:

> *"Der Unterricht muss dann so angelegt werden, dass sich diese - wenigen - fundamentalen Ideen bei den Schülerinnen und Schülern bilden können, er muss Kenntnisse und Erfahrungen vermitteln, die anhand dieser Ideen zu ordnen sind, und er muss diese Ideen zu einem geeigneten Zeitpunkt explizit thematisieren, um die spezifischen Möglichkeiten und Beschränkungen der Informatik in Abgrenzung gegen andere Disziplinen erkennbar zu machen."*

Einige Beispiele fundamentaler Ideen und die dazugehörigen Kriterien sind in [Sch94] zu finden.

2 Programmiersprachen im Unterricht

Ein vieldiskutiertes Thema ist, welche Sprache sich für den Unterricht am besten eignet. [LGS06, Pal90] meint, dass der Lernerfolg von der Zeit abhängt, die für tatsächliches Programmieren aufgewendet werden kann. Es sollte keine Zeit mit sprachspezifischen Eigenheiten und Syntaxproblemen "verschwendet" werden. Geht es nach [LGS06, Mil93], hat eine Programmiersprache für Unterrichtszwecke einen einfachen Zugang. Sie sollte leicht erlernbar sein, eine klare Struktur haben und vielfältig einsetzbar sein. Die Sprache hat eine einfache Syntax, einfaches I/O

Handling, verständliche String-Manipulation, aussagekräftige Schlüsselwörter und verständliches Feedback im Fehlerfall.

Während Pascal und Logo vor einigen Jahren noch oft im Unterricht zum Einsatz kamen, sind beide heute stark aus der Mode gekommen. Gründe dafür sind sicher die mangelnde Einsatzfähigkeit in der Industrie und die, so gut wie nicht mögliche, Verwendbarkeit der Sprache bei steigender Komplexität der Software.

Heute zählen Sprachen wie C, Java und C++ zu den beliebtesten Programmiersprachen. Das ist an der Anzahl der verfügbaren Entwickler, Lehrgänge und Dienstleister weltweit zu erkennen [TIO07]. Studien wie [dWT02], [Md03] und [SW98] belegen, dass Java, C++ und C die Sprachen mit der größten Verbreitung an Universitäten sind.

Trotz dieser Beliebtheit (oder gerade deswegen) wird viel über die Tauglichkeit dieser Sprachen im Unterricht diskutiert, gerade wenn es um geeignete Sprachen für Programmieranfänger geht. Die oben genannten Sprachen werden als überladen betrachtet. Die Studierenden plagen sich eher mit der Notation, als mit dem tatsächlichen Algorithmen. [LGS06] erläutert, dass die meisten Probleme von Programmierneulingen immer dasselbe Muster zeigen:

> *"[...] construct-based problems, which make it difficult to learn the correct semantics of language constructs, and plan composition problems, which make it difficult to put plans together correctly [...] students lack the skills needed to trace the execution of short pieces of code after taken their first course on programming."*

Im Zuge einer Studie an der finnischen Universität in Tampere ist eine Umfrage an europäischen Hochschulen durchgeführt worden. 559 Studenten und 34 Lektoren an 6 verschiedenen Universitäten wurden unter anderem zu deren Schwierigkeiten beim Lernen und Lehren von Programmiersprachen befragt. Die daraus abgeleiteten Folgerungen besagen:

> *"the most difficult concepts to learn are the ones that require understanding larger entities of the program instead of just details [...] abstract concepts like pointers and memory handling are difficult to learn [...] However, the biggest problem of novice programmers does not seem to be the understanding of basic concepts but rather learning to apply them."* [LAMJ05]

Die Studenten haben Probleme, wenn es darum geht, den Gesamtumfang eines Programmes zu erfassen und umzusetzen, dh. wie setze ich eine mir gestellte Aufgabe mit den Konzepten um, die ich gelernt habe. Dabei darf die Syntax einer Sprache nicht im Wege stehen, da sie daran hindert eine Problemlösung zu finden und nur neue, andere Probleme schafft. Zeiger und Speichermanipulation zählen zu den als sehr schwierig eingestuften Konzepten.

Die in diesem Kapitel zitierte Literatur deckt sich mit den Beobachtungen des Autors aus der eigenen Ausbildung und der Abhaltung eines C Tutoriums für Programmieranfänger. Die Studenten konnten genau wiedergeben, wie sie das Problem lösen würden, doch konnten sie es nicht "zu Papier" bringen, also als Quelltext wiedergeben. Die Syntax wurde als unnatürlich und teilweise unverständlich empfunden. Manche syntaktische Eigenheiten sind für den Tutor auch schwer zu erklären, da die Studenten die dahinterliegenden Konzepte noch nicht verstehen können. Nahezu alle Fehler resultierten aus einem Fehler in der Syntax. Der Autor konnte dabei ein hohes Maß an Frustration und Demotivation der Studenten im Unterricht feststellen. Die Sinnhaftigkeit des Lehrinhaltes wurde weiters angezweifelt.

3 Python als erste Programmiersprache?

Warum eignet sich Python als Werkzeug für den Informatikunterricht? Ist Python als Einstiegssprache geeignet? Was zeichnet Python gegenüber anderen Programmiersprachen aus? Welche Anforderungen werden an eine "erste Programmiersprache" gestellt? Was haben die Lehrenden davon? Dieses Kapitel enthält eine Diskussion zu den vorangegangen Fragen.

Gegenwärtig sind hauptsächlich Java und C++ als Programmierprachen in Lehrgängen für Programmieranfänger anzutreffen (vgl. Kapitel 2) [Rad06].

> *"While students with good preliminary background, who have already committed themselves to a computing career, usually succeed in such courses, many others remain disappointed or even completely fail."*

[Rad06] schreibt, dass Studenten mit Vorkenntnissen und explizitem Interesse an der Materie es leichter haben, solche Kurse erfolgreich zu absolvieren. Studenten, die bisher keine Erfahrungen in der Programmierung haben, bleiben hier oft auf der Strecke. Die Motivation, die Materie weiter zu verfolgen, ist verständlicherweise gering. Dabei soll Unterricht üblicherweise motivieren und dadurch den Forschergeist wecken. Ob das gelingt, ist eine Frage der Gestaltung des Unterrichts und hängt nur geringfügig von der zu unterichtenden Materie ab. Jedes noch so trockene Thema kann durch interessante und fesselnde Unterrichtsgestaltung schmackhaft gemacht werden. Die Unterrichtsgestaltung hängt auch mit der Wahl der benutzten Werkzeuge, um Wissen zu vermitteln, zusammen.

Das Problem nach [Rad06] ist, dass die heute verwendeten Sprachen alle kommerziell ausgerichtet sind. Kommerzielle Sprachen sind nicht für den Unterricht ausgelegt, sondern für die industrielle Softwareentwicklung. Kommerzielle Applikationen sind komplex. Die Programmiersprachen sind dafür ausgelegt, die Entwicklung dieser Applikationen zu ermöglichen.

Der Autor unterstützt diese Argumentation. Es stellt sich jedoch die Frage, warum nicht gerade aus diesen Gründen Programmiersprachen so einfach und übersichtlich wie möglich sein sollten. Eine Ursache dafür ist sicher eine versuchte Effizienzsteigerung durch diverse Sprachkonstrukte, die den Profi effizienter arbeiten lassen, für Anfänger aber nicht leicht verständlich sind ("make the common case efficient"). [Rad06] meint in seiner Arbeit, dass genau aus diesem Grund Programmierneulinge mit Konzepten konfrontiert werden, die für den Anfang überfordern.

[BM05] bringen einen Vergleich für den sportlichen Leser, indem sie ein Beispiel aus dem Schifahren bringen. Um ein wirklich guter Schifahrer zu werden, erfordert es Übung. Schipisten sind darauf ausgelegt, das Üben zu unterstützen, indem sie je nach Schwierigkeitsgrad farblich markiert sind. Blau für Anfänger, Rot für Fortgeschrittene und Schwarz für Profis. Eine blaue Piste bedeutet aber nicht, das hier auf die wichtigen Elemente des Schifahrens zu verzichten ist. Schwünge, Geschwindigkeitskontrolle, Bremsen und die richtige Übersicht sind auch hier notwendig, sind jedoch unter entschärften Bedingungen ausübbar. Wird ein Anfänger auf die schwarze Piste geschickt, wird er oft stürzen. Die Erfahrung ist für ihn frustrierend, und er wird schnell aufgeben und den Sport verdammen.

Dieser Ausflug in eine anderes Gebiet und die einfache Erkenntnis daraus kann auf jedes Lernen umgelegt werden. Konfrontiert man den Lernenden zu früh mit zu hoher Komplexität, kann dieser nicht genug Erfolge erleben und wird sich eher frustriert von der Thematik abwenden. Der Stoff

muss sich aufbauend an den fundamentalen Ideen des Unterrichtsgegenstandes nähern, und dem Lernenden zwischendurch immer wieder Erfolgserlebnisse ermöglichen.

> *"As students progress through the introductory computer science sequence, we want them to focus on aspects of problem solving, algorithm development, and algorithm understanding. Unfortunately, many modern programming languages require that students jump into more advanced programming concepts at a point that is too soon in their development. This sets them up for possible failure, not because of the computer science but because of the language vehicle being used."* [BM05]

In der Informatik liegt das Augenmerk auf den Aspekten der Problemlösung, der Entwicklung und dem Verstehen von Algorithmen. Die heute im Unterricht angewandten Programmiersprachen führen aber zu früh in komplexere Details der Wissenschaft. Das führt auch dazu, dass nicht Informatik, sondern die Aspekte einer Programmiersprache zum Gegenstand des Unterichts werden.

> *"It is a shame that the languages that our students encounter when they come to learn to program are languages that are designed for commercial use by experienced commercial programmers. Modern languages like Java and C++ contain a host of features that most students will never need, and which we should probably admit are a mystery to many of the students' teachers."* [Jen03]

Sprachen wie Java oder C++ sind in ihrem Umfang so mächtig, das es bestimmt eher selten ist, wirklich routinierte Profis als Lektoren zu bekommen. Glücklich sind diejenigen, die in diesen Genuss kommen.

Industriesprachen

Weßhalb werden Java, C++ und C nun hauptsächlich verwendet? Geht es nach [Jen03], [LGS06], [Zel98], [LM06], [Md03] und [Sta00], ist dieser Zustand industriegetrieben. Studenten sehen Jobangebote, in denen eine bestimmte Programmiersprache als Voraussetzung verlangt wird, und wollen diese lernen, um ihre Qualifikation damit aufzuwerten. Sie haben kein Interesse, eine "Baby-Sprache"[PM06] zu lernen. Lehreinrichtungen bewerten ihre Studienpläne anhand der Nachfrage in der Industrie. Das betrifft ganz besonders Fachhochschulen, wo Lehrpläne in enger Zusammenarbeit mit der Industrie erstellt werden.

Ein Lektor des Autors, der lieber anonym bleiben möchte, ist da ganz anderer Meinung. Die obig gebrachten Überlegungen seien unwahr, vielmehr gestaltet sich der Unterricht oft nach der Verfügbarkeit von Lektoren im Bekanntenkreis der für den Studienplan verantwortlichen Personen. Diese Lektoren bringen eher ihre persönlich bevorzugten Themen und Werkzeuge in den Unterricht ein (wobei letzteres durchaus legitim und wichtig für die Qualität des gebrachten Unterichtsstoffes ist).

Nach Ansicht des Autors ist die Realität wohl eine Mischung aus beiden gerade genannten Punkten. Doch sollte es nicht anders sein? In der Informatik war es früher auch anders. Blickt man auf die Zeit vor der objektorientierten Bewegung zurück, gab es Pascal [Wir71] als allgemein anerkannte Programmiersprache für den Unterricht [Rad06],[Zel98]. Das Problem mit reinen Unterrichtssprachen wie Pascal ist jedoch, dass sie in der Industrie nicht verwendet werden. Aus diesem Grund wurde Pascal auch weltweit aus den Lehrplänen gestrichen.

Es braucht also eine Sprache, die

- in der Industrie anerkannt ist bzw. in der Industrie Verwendung findet,
- mit der komplexe Softwareprojekte umgesetzt werden können,
- die für den Unterrichtseinsatz geeignet ist.

Betrachten wir Python, sind die ersten beiden Punkte erfüllt. Wenden wir uns nun dem dritten Punkt zu. Ist Python für den Unterrichtseinsatz geeignet?

Hello World

Hello World ist traditionellerweise das erste Programm jedes Programmieranfängers. Starten wir gleich mit der Python Variante.

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

Listing 1: Hello World mit Python

Nun zur C++ Variante des Programms.

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main ()
6 {
7   cout << "Hello World!";
8   return 0;
9 }
```

Listing 2: Hello World mit C++

- Zeile 1: Zeilen, die mit einer Raute(#) beginnen, sind Direktiven für den Präprozessor. Diese Zeilen werden vom Compiler nicht normal ausgewertet, sondern zuvor vom Präprozessor bearbeitet. Dieser reagiert je nach Befehl unterschiedlich. In diesem Beispiel wird dem Präprozessor mitgeteilt, die Datei `iostream` zu inkludieren. Diese enthält Deklarationen der Standard Eingabe-Ausgabe Bibliothek von C++. Sie wird benötigt, da wir einen Befehl aus dieser Bibliothek später im Programm verwenden.
- Zeile 3: Alle Elemente der Standard C++ Bibliothek werden in einem eigenen Namensraum deklariert. Dieser Namensraum hat den Namen `std`.
- Zeile 5: Die `main` Funktion ist der Startpunkt eines jeden C++ Programmes. Eine Funktion wird immer mit runden Klammern definiert. Innerhalb der runden Klammern können Parameter an die Funktion übergeben werden. Die geschwungenen Klammern markieren den Bereich der Funktion. Alles innerhalb dieser Klammer wird beim Aufruf der Funktion ausgeführt. Vor dem Namen der Funktion steht die Definition des Rückgabewertes.
- Zeile 7: Das ist ein C++ Befehl (statement), den wir über eine Library inkludiert haben und einem Namesraum zugewiesen haben. Dieser spezielle Befehl repräsentiert den *Standard Output Stream* in C++. Das Beispiel zeigt, wie eine Sequenz von Zeichen (“Hello World!“) in den Stream geschrieben wird. Jeder Befehl muss mit einem Semikolon (;) beendet werden.

- Zeile 8: Der `return` Befehl beendet die Funktion, der Returnwert muss vom gleichen Typ wie die Deklaration der Funktion sein. Der Returncode von 0 bedeutet, das die Funktion fehlerfrei ausgeführt wurde.

Wie zu sehen ist, müssen, bei diesem sehr simplen Beispiel, Themen, die für einen Anfänger doch sehr fortgeschritten sind, erläutert werden. Eine andere, und nach Erfahrung des Autors sehr beliebte Variante um diese Problematik zu umschiffen, ist das gezielte Ignorieren dieser Punkte. Der Vortragende ersucht die Studenten, gewisse Teile des Programmes einfach auszublenden und z.B. nur innerhalb der `main` Funktion zu arbeiten und zu denken.

> *"The C++ version has always forced me to choose between two unsatisfying options: either to explain the #include, void main(), {, and } statements, and risk confusing or intimidating some of the students right at the start, or to tell them 'just don't worry about all of that stuff now, we will talk about it later' and risk the same thing."*[Elk00]

Keine der beiden Varianten ist befriedigend. [Elk00] beschreibt den Aufwand, ein *Hello World* Programm in ein Skriptum zu verfassen, wie folgt.

> *"There are thirteen paragraphs of explanation of "Hello, world" in the C++ version, in the Python version there are only three. More importantly, the missing ten paragraphs do not deal with the "big ideas" in computer programming, but with the minutia of C++ syntax. I found this same thing happening throughout the chapters that I have completed so far. Whole paragraphs simply disappear from the Python version of the text because Python's much simpler syntax renders them unnecessary."*

[Elk00] zeigt damit, dass viel Einführungsarbeit mit C++ auf Spracheigenheiten bezogen ist. Das ist Zeitverschwendung. Studenten könnten mit Python viel früher zu ersten Erfolgserlebnissen kommen, ohne über mysteriöse Gegebenheiten, wie unerklärte und nicht verstandene Konstrukte, in den eigenen Programmen zu stolpern.

Bei der Java Version des Programmes wird der Student sofort mit einem fortgeschrittenen Konzept konfrontiert, welches aber das Wissen über darunterliegende, objektorientierte Thematik voraussetzt.

Der Autor will C++ und Java nicht negativ darstellen. Es soll lediglich aufgezeigt werden, dass ein Start in die Programmierung mit einer der momentan in der Ausbildung verwendeten Sprachen nicht als optimal angesehen werden kann. Ein großer Vorteil von Python im Vergleich der *Hello World* Programme, ist die unterschiedliche Anzahl der Konzepte, die verstanden werden müssen, um ein erstes Programm zu schreiben. Ein erstes funktionales Programm ist auch ein erstes Erfolgserlebnis und motiviert zu mehr.

Syntax

> *"Compared to languages such as Java or C++, Python has a more intuitive syntax. Python enforces an intended and structured way of writing programs, and the code resembles pseudo code."*[LGS06]

Entwickler werden unter Python gezwungen, mit Einrückung ihres Quelltextes zu arbeiten. Eine Einrückung markiert einen Block im Programm. Der Block endet erst mit der Einrückung. Bei

falscher Einrückung wird ein Fehler generiert. Dabei sieht ein Programm in Python fast genauso aus wie das Programm geschrieben in Pseudocode. Das hat den Vorteil, dass Algorithmen einfach nach Pseudocodevorlage implementiert werden können.

> *"[...] with Python, the Pseudocode is in fact also real code, and already executable."*

[Sta00] meint sogar, dass mit Python sei Pseudocode auch echter Quelltext, der schon ausführbar ist.

Die Einrückung zwingt dazu, strukturiert zu schreiben und fördert dadurch ein einheitliches Programmbild, während in anderen Sprachen verschiedene Schreibstile möglich sind, die dazu führen, dass Richtlinien zur Lesbarkeit entwickelt werden müssen.

Programme werden lesbarer, was einige Vorteile mit sich bringt. Lektoren haben es leichter, die Arbeit der Studenten zu evaluieren und finden auch schneller Fehler in Programmen. Da Python keine unterschiedlichen syntaktischen Stile erlaubt, ist es somit für Studenten einfacher, an größeren Projekten gemeinsam zu arbeiten.[Elk00]

> *"Novice programmers find indentation to be quite natural and we have found that it enforces a more readable coding style than students typically adopt when they can use braces."*[BM05]

Listing 3 zeigt ein C++ Beispiel. Frei nach dem Auge würde der zweite Befehl zum if-Block gehören. Durch das Fehlen der geschwungen Klammern umfasst der Block jedoch nur den ersten Befehl, der zweite wird unabhängig von der Verzweigung ausgeführt. Dieser Fehler passiert recht häufig. Mit Python wäre dieses Beispiel klar, alles Eingerückte nach der Verzweigung ist dem Block zugehörig. Es herrscht eine Art "natürliche Lesbarkeit".

```
1 if(t == 1)
2     cout << "t is 1";
3     t = 2;
```

Listing 3: Beispiel zur Lesbarkeit von eingerücktem Quelltext

> *"We feel that writing structured programs, which are easy to check, follow and maintain, is one of the main lessons in any programming course. Using Python, this lesson is taught automatically."*[LGS06]

Lesbaren Quelltext zu schreiben ist ein wichtiges didaktisches Ziel. Durch Python erreicht der Student das implizit.

Semantik

Durch die dynamische Typisierung erspart man sich das Konzept der Deklaration. Der Typ einer Variable wird zur Laufzeit bestimmt. Eine Variable zeigt immer auf ein Objekt. Dieses Konzept ist leichter zu erläutern und zu verstehen, als die maschinennahe statische Typisierung, wo eine Variable ein Platz im Speicher ist, der zuvor deklariert werden muss. Typisierung ist ebenfalls eine häufige Fehlerquelle, auf die anfangs getrost verzichtet werden kann. Auch Zeiger und Speicherverwaltung sind solche Konzepte.

Die Literatur ist sich einig, dass die genannten Konzepte für eine erste Programmiersprache nicht notwendig sind, aber trotzdem zu einem späteren Zeitpunkt gelehrt werden sollten. Erfahrungen zeigen, dass die Konzepte auch viel einfacher angenommen werden, nachdem Studenten ihre ersten Erfolge mit Python erlebt haben und in der Lage sind, eigene komplexe Programme zu schreiben. Der Umstieg auf eine Hochsprache ist unproblematisch.[Jen03, Sha03, Rad06, Zel98, Elk00, PM06]

> *"We believe that it is advantageous for beginning students to spend time on the green runs[blaue Piste] learning the rudimentary ideas relating to algorithms and data structures. If you are among those of our colleagues who are frustrated with Java or C++, we encourage you to consider Python and join us on the green runs, it works."*

[Elk00] nimmt wieder das Beispiel des Schifahrens als Illustration und vergleicht Python als erste Programmiersprache mit dem Fahren auf einer blauen Piste, wo Studenten, unter entschärften Bedingungen, das Arbeiten mit Algorithmen und Datenstrukturen lernen.

4 Programmierparadigmen im Unterricht

Python ist primär eine objektortientierte Sprache, unterstützt also das objektorientierte Softwareparadigma. Jedoch ist der Entwickler bei der Verwendung von Python an kein spezielles Paradigma gebunden. Python kann somit als **Multiparadigmen-Sprache** bezeichnet werden.

Nach [Pla93] und [Wes99] sind Multiparadigmensprachen für den Unterricht gut geeignet. Es ist wichtig den Studenten in kein Korsett hineinzuzwingen, sondern ein breites Schema an Möglichkeiten zur kreativen Entfaltung anzubieten, denn

> *"verschiedene Menschen nehmen zur Lösung von gleichen Aufgaben unterschiedliche Sichtweisen ein, und ebenso kann es sein, dass ein Mensch für unterschiedliche Aufgaben verschiedene Lösungsansätze verfolgt. Für die wenigsten Probleme gibt es die optimale Lösung, und so empfiehlt es sich, der- oder demjenigen, der das Problem löst, die Wahl zu überlassen, wie er oder sie am besten mit einer Aufgabe umgeht. Daher ist es für den praktischen Einsatz von Programmiersprachen notwendig, diese unterschiedlichen Sichtweisen, die zu verschiedenen Lösungsstrategien führen, in geeigneter Form ausdrücken zu können."*[Gra03]

Das bietet den Studenten den Freiraum, zu Beginn ihrer Programmierkarriere mittels **eines** Werkzeugs, nämlich der Programmiersprache, unterschiedliche Denkweisen und Programmieransätze zu erproben. Die Studenten können sich voll und ganz auf das Verstehen der, den Paradigmen zugrundeliegenden Prinzipien konzentrieren und begreifen, dass ein Paradigma nicht von der verwendeten Programmiersprache abhängt, sondern von der Sichtweise auf eine Problemstellung und deren Lösung.

5 Algorithmen

Das Erlernen von grundlegenden Algorithmen ist eines der wichtigsten fachdidaktischen Ziele. Es ist nicht nur ein fachübergreifendes Konzept, sondern auch ein essentielles Werkzeug, um komple-

xe Problemstellungen lösen zu können. Dabei ist es wichtig, den Studierenden einen Algorithmus auch ausprobieren zu lassen. Das einfache Erläutern mit natürlicher Sprache und Pseudocode, ist zwar als Unterstützung und erste Beschreibung notwendig, hat aber nicht den entsprechenden Lerneffekt. Es muss eine Programmiersprache als Hilfsmittel angewandt werden. Dazu eignet sich natürlich Python, da man sich durch die einfache und Pseudocode-ähnliche Syntax mehr auf die Implementierung und das Verstehen des Algorithmus konzentrieren kann, als auf das Verstehen der Programmiersprache. Diese Aussage wird durch nachfolgendes Zitat aus [Cho02] gestützt.

> *"Design and analysis of algorithms are a fundamental topic in computer science and engineering education. Many algorithms courses include programming assignments to help students better understand the algorithms. Unfortunately, the use of traditional programming languages forces students to deal with details of data structures and supporting routines, rather than algorithm design. Python represents an algorithm-oriented language that has been sorely needed in education. The advantages of Python include its textbook-like syntax and interactivity that encourages experimentation."*

[Cho02] spricht sogar von einer *algorithmus-ortientierten* Sprache, wenngleich dieser Ausdruck etwas übertrieben erscheint. Jeder Algorithmus lässt sich in jeder beliebigen Programmierprache implementieren und zeigt die jeweiligen verschiedenen Lösungswege eines Algorithmus auf. [Cho02] scheint damit einfach die Eignung Pythons für die praktische Anwendung von Algorithmen im Unterricht unterstreichen zu wollen.

6 Zusammenfassung

Für Anfänger ist das Erlernen einer Programmiersprache schwierig. Das liegt daran, dass im heutigen Unterricht bevorzugt mit Sprachen aus der Industrie gelehrt wird. Studenten wollen in der Industrie Jobs bekommen, und legen deshalb Wert darauf, dass gefragte Technologien Bestandteil ihrer Ausbildung sind. Die Industrie wiederum will ihren Bedarf befriedigen. Dabei wird übersehen, dass Programmiersprachen keine Technologien selbst, sondern Werkzeuge für Technologien sind. Im Unterricht muss eine Programmiersprache ein Werkzeug sein, mit dessen Hilfe es möglich ist, die fundamentalen Ideen eines Unterrichtsgegenstandes zu vermitteln, ohne in einen Unterricht über die Programmiersprache selbst abzudriften.

Die vorliegende Arbeit stellt Python als ein solches Werkzeug vor. Sie zeigt auf, dass Python, im Gegensatz zu heute häufig im Unterricht zum Einsatz kommenden Sprachen (wie C, C++ oder Java), gut für Anfänger geeignet ist. Aufgrund des einfachen Zugangs können mit Python viel früher relevante Konzepte der Informatik und Softwareentwicklung diskutiert werden. Ein weiterer wesentlicher Vorteil ist, dass auch die Arbeit der Unterrichtenden erleichtert wird. Die Sprache ist kompakt und simpel gehalten und versucht sich dem Entwickler nicht in den Weg zu stellen. Gleichzeitig ist sie eine allgemein anerkannte Sprache und findet Verwendung in der Industrie.

Literatur

[BM05] BN MILLER, DL RANUM: *Teaching an Introductory Computer Science Sequence with Python*. Technischer Bericht, Luther College, Iowa, 2005.

[Bru60] BRUNER, JEROME: *The Process of Education*. Harvard University Press; New Edition (November 1, 2006), 1960.

[Cho02] CHOU, P. H.: *Algorithm Education in Python*. In: *Proceedings of Python 10*, 2002.

[dWT02] DERAADT, M., R. WATSON und M. TOLEMAN: *Language Trends in Introductionary Programming Courses*. Informing Science and IT Education Conference, 2002.

[Elk00] ELKNER, JEFFREY: *Using Python in a High School Computer Science Program*. Technischer Bericht, Yorktown High School, Arlington, Virginia, 2000.

[Gra03] GRABMÜLLER, MARTIN: *Multiparadigmen-Programmiersprachen*. Technischer Bericht, Technische Universität Berlin, 2003.

[Jen03] JENKINS, TONY: *The First Language - A Case for Python?* In: *4th Annual LTSN-ICS Conference (Galway)*, Seite 7, 2003.

[LAMJ05] LAHTINEN, ESSI, KIRSTI ALA-MUTKA und HANNU-MATTI JÄRVINEN: *A study of the difficulties of novice programmers*. In: *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, Seiten 14–18, New York, NY, USA, 2005. ACM Press.

[LGS06] LINDA GRANDELL, MIA PELTOMAKI, RALPH-JOHAN BACK und TAPIO SALAKOSKI: *Why complicate things?: introducing programming in high school using Python*. In: *ACE '06: Proceedings of the 8th Australian conference on Computing education*, Seiten 71–80, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[LM06] LINDA MANNILA, MICHAEL DERAADT: *An Objective Comparison of Languages for Teaching Introductory Programming*. Technischer Bericht, Åbo Akademi University, University of Southern Queensland, 2006.

[Md03] M. DERAADT, R. WATSON: *Language Tug-Of-War: Industry Demand and Academic Choice*. Technischer Bericht, University of Southern Queensland, 2003.

[Mil93] MILLBRANDT, G.: *Using problem solving to teach a programming language in computer studies*. Journal Of Computer Science Education, 8(2), 1993.

[Mod02] MODROW, ECKART: *Pragmatischer Konstruktivismus und fundamentale Ideen als Leitlinien der Curriculumentwicklung am Beispiel der theoretischen und technischen Informatik*. Doktorarbeit, Martin-Luther-Universität Halle-Wittenberg, 2002.

[Pal90] PALUMBO, DAVID B.: *Programming Language/Problem-Solving Research: A Review of Relevant Issues*. Review of Educational Research, 60(1), 1990.

[Pla93] PLACER, JOHN: *The promise of multiparadigm languages as pedagogical tools*. In: *CSC '93: Proceedings of the 1993 ACM conference on Computer science*, Seiten 81–86, 1993.

[PM06] PATTERSON-MCNEILL, HOLLY: *Experience: from C++ to Python in 3 easy steps*. J. Comput. Small Coll., 22(2):92–96, 2006.

[Rad06] RADENSKI, ATANAS: *"Python first": a lab-based digital introduction to computer science*. In: *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, Seiten 197–201, New York, NY, USA, 2006. ACM Press.

[Sch93] SCHWILL, ANDREAS: *Fundamentale Ideen der Informatik*. Technischer Bericht, Universität Oldenburg, 1993.

[Sch94] SCHWILL, ANDREAS: *Fundamentale Ideen: Eine Studie zur Methodologie der Informatik*. Technischer Bericht, Universität Paderborn, 1994.

[Sha03] SHANNON, CHRISTINE: *Another breadth-first approach to CS I using python*. SIGCSE Bull., 35(1):248–251, 2003.

[Sta00] STAJANO, FRANK: *Python in Education: Raising a Generation of Native Speakers*. Technischer Bericht, University of Cambridge, 2000.

[SW98] STEPHENSON, C. und T. WEST: *Language choice and key concepts in cs 1*. Journal of Research on Computing Education, 31(1), 1998.

[TIO07] *TIOBE Programming Community Index*. `http://www.tiobe.com/tpci.htm`, 2007. [Online; Stand 21. November 2007].

[Wes99] WESTBROOK, D.S.: *A multiparadigm language approach to teaching principles of-programming languages*. In: *Frontiers in Education Conference, 1999. FIE '99. 29th Annual*, Seiten 11B3/14–11B3/18, 1999.

[Wir71] WIRTH, NIKLAUS: *The programming language pascal*. Acta Informatica, 1(1):35–63, 1971.

[Zel98] ZELLE, JOHN M.: *Python as a First Language*. Technischer Bericht, Wartburg College, 1998.