FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Programmiereinstieg durch App-Entwicklung

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Eldin Ducic
Matrikelnummer 0405503

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao.Univ.Prof. Dipl.-Ing. Dr.techn Gerald Futschek

Wien, 10.11.2015 _____ _____

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Eldin Ducic
Hoffingergasse 12-14/3/1, 1120 Wien

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, 10.11.2015 _____

Abstract

Programing is seen as a complex discipline, frequently associated with difficulties when being lerned. The complicity of the new concepts, which require new patterns of thought, often cause disinterest and loss of motivation for learning. This thesis suggests an apprach to teach programming by developing apps in order to foster motivation through the use of new technology and application oriented examples. It delivers an answer to the research question how an introduction to programming through app development could be designed and which assignment of tasks would be suitable for this purpose.

After determining through literature research which concepts are to be included in an introductory course and identifying the barriers for novices to programming, a prototype curriculum with suitable tasks and exercises is accordingly designed and subsequently evaluated based on guidelines from literature and argumentation. The main outcome of the thesis is an introductory programming curriculum with examples and exercises adapted for app-development. Beginners are enabled to create simple apps on their own, which should increase the interest and motivation for learning how to program.

Kurzfassung

Programmieren gilt als eine komplexe Disziplin, deren Erlernen häufig mit Schwierigkeiten verbunden ist. Die Kompliziertheit der Konzepte, die eine Umstellung auf neue Denkmuster erfordern, geht oft mit Desinteresse und Verlust an Motivation zum Lernen einher. Diese Arbeit schlägt den Ansatz vor, Programmierkenntnisse durch Entwicklung von Apps zu vermitteln, um anhand der Verwendung von neuen Technologien und anwendungsnahen Programmierbeispielen motivationsfördernde Effekte zu erzielen. Sie gibt eine Antwort auf die Frage, wie der Einstieg ins Programmieren durch App-Entwicklung gestaltet werden kann und was für Aufgabenstellungen dafür geeignet sind.

Dafür werden zunächst durch Literaturrecherche die Basiskonzepte des Programmierens festgelegt, sowie die häufigsten Hürden für Novizen identifiziert. Darauf aufbauend wird ein Prototyp eines Curriculums mit passenden Aufgaben didaktisch aufbereitet und anschließend anhand von Richtlinien aus einschlägiger Literatur und Argumentation evaluiert.

Das zentrale Ergebnis dieser Arbeit ist ein Curriculum mit entsprechenden Unterrichtsszenarien für Programmiereinsteiger, welche grundlegende Konzepte des Programmierens durch App-Entwicklung vermitteln. Novizen werden befähigt, einfache Anwendungs-Apps selbst zu erstellen, wodurch das Interesse und Motivation am Programmierunterricht gesteigert werden soll.

Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabenstellung

Programmieren ist eines der Teilgebiete der Informatik, das nach wie vor für Einsteiger als sehr schwer bewältigbar gilt. Es gibt mehrere Hürden, auf die Novizen in diesem Fach oft stoßen. Unter anderem wirkt die Komplexität und der Bedarf an besonderen Abstraktionsfähigkeiten ziemlich abschreckend. Man wird mit vielen Konzepten, wie z.B. Schleifen, Parameter, Arrays, etc., zum ersten mal konfrontiert, was sich als schwierig erweist (Kölling & Rosenberg, 2001). Des Weiteren könnte von einem sehr langen Weg ausgegangen werden, um halbwegs verwendbares Können zum „richtigen" Programmieren zu erlangen, dessen Resultat konkrete Anwendung im Leben des jeweiligen Schülers/Schülerin bzw. Student/Studentin finden kann. Bis dorthin könnten die vorgestellten Programmierkonzepte oft verwirrend und realitätsfremd wirken, was nicht sehr stimulierend ist. Viele, die Programmieren als Pflichtfach haben, versuchen es irgendwie mit zumindest genügender Leistung zu absolvieren und es so schnell wie möglich hinter sich bringen.

Ein neuer Impuls beim Programmierunterricht wäre angebracht, der eine naheliegendere und „greifbare" Herangehensweise anbietet und somit in der Wissensvermittlung bei Anfängern mehr Interesse weckt. Diese Arbeit schlägt in dieser Hinsicht einen etwas anderen Weg zum Erlernen des Programmierens vor, als er derzeit üblich ist, und zwar durch App-Entwicklung. Es wird versucht, Smartphones im Programmierlernprozess auf einer höheren Ebene (Coding) zu verwenden, um die Realitätsnähe von den jeweiligen Programmierkonzepten anhand erstellter Apps zu verdeutlichen und somit die Lernmotivation zu steigern. Aufgrund der spezifischen Rahmenbedingungen wäre der Ansatz für eine Zielgruppe ab dem Oberstufenalter (ca. 14 Jahre) geeignet.

1.2 Fragestellung

In der Regel wird davon ausgegangen, dass bereits ausreichende Programmierkenntnisse vorhanden sein müssen, um auf die App-Entwicklung umzusteigen. Diese könnte jedoch aus Motivationsgründen als Basis statt wie bisher als Überbau betrachtet werden. Plausible Argumente dafür könnten davon abgeleitet werden, dass die meisten Schüler heutzutage über ein Smartphone mit etlichen darauf installierten Apps verfügen. Apps sind von der Implementierung her relativ kleine, jedoch oft sehr

nützliche Programme, die sehr beliebt unter der jungen Population sind.
Vom didaktischen Aspekt der Programmierung her sind sie aber vor allem etwas konkretes, für den Schüler greifbares, womit er seine erbrachte Leistung in ein populäres Produkt mit unmittelbarem Nutzen umgewandelt betrachten und Freunden und Bekannten vorzeigen kann. Bereits das einfachste „Hello world“ Programm dürfte als App mehr Gewicht haben und subjektiv als eine größere Errungenschaft gesehen werden als die Ausgabe in der Konsole an einem Desktop Rechner. Dieser Effekt sollte ausgenutzt werden, um mehr Interesse am Programmierunterricht zu wecken und Anfänger in dieser Hinsicht zu begeistern.

Die Forschungsfrage dieser Arbeit würde sich daher wie folgt stellen:

Wie kann der Einstieg ins Programmieren für Anfänger durch App-Entwicklung gestaltet werden?

Als Unterfrage würde sich ergeben: Wie könnte ein Curriculum für den Einstieg in das Programmieren durch App-Entwicklung gestaltet sein?

Die zweite Unterfrage wäre: Welche Aufgabestellungen für Apps wären geeignet zum Erklären von Programmierkonzepten durch diesen Ansatz, sodass sie für die Studierenden verständlich und begreifbar sind, sowie Interesse durch Realitätsnähe erwecken?

1.3 Ziel der Arbeit

Das Ziel dieser Arbeit ist es, ein Curriculum mit entsprechenden Unterrichtsszenarien zu erarbeiten, das auf dem Ansatz basiert, Programmieren durch App-Entwicklung zu erlernen. Unter Beachtung didaktischer Aspekte sollen die zu erlernenden Programmierkonzepte durch Beispiele, die direkt auf Smartphones bzw. einem Emulator ausprobiert werden sollen, der jeweiligen Zielgruppe von SchülerInnen / Studierenden präsentiert werden.

Ein darauf aufbauendes Ziel ist die Implementierung von einfachen Apps, die alle, oder zumindest die meisten, bearbeiteten Konzepte enthalten. Der Effekt eines greifbaren „Fertigprodukts“ soll dabei als Hauptmotivation zum Programmierunterricht dienen. Darüber hinaus könnte noch die Veröffentlichung und Onlinestellung eines oder mehrerer Apps als Zusatzstimulus in Erwägung gezogen werden.

1.4 Methoden und Vorgehensweise

Um das vordefinierte Ziel dieser Diplomarbeit zu erreichen wird wie folgt vorgegangen:

- Im Theorie-Teil erfolgt zunächst eine Literaturrecherche und die Identifizierung der Problembereiche, die für Programmieranfänger am häufigsten als Hürde festgestellt werden. Des Weiteren werden gegenwärtige didaktische Ansätze im Programmierunterricht erläutert, sowie die wichtigsten Konzepte, die zu erlernen sind, dargeboten.
- Im praktischen Teil der Arbeit wird ein Curriculum konzipiert, das die Bedürfnisse und kognitive Kapazitäten von Programmieranfängern berücksichtigt und auf die technischen Rahmenbedingungen von mobilen Geräten zugeschnitten ist. Darauf aufbauend werden Unterrichtsszenarien vorgeschlagen, um grundlegende Konzepte des Programmierens zu vermitteln. In möglichst kleinen Schritten werden diese vorgestellt und anschließend anhand von praktischen Problemstellungen umgesetzt. Durch schrittweises Vorgehen werden die Konzepte erweitert bis ausreichend Kenntnisse vorhanden sind, simple Apps mit einfachen Funktionalitäten zu entwickeln. Dadurch sollte unmittelbar ersichtlich sein, welchen praktischen Nutzen die erlernten Konzepte haben können und wie sie in der Realität Anwendung finden.
- Im Anschluss erfolgt eine Evaluation des Curriculums und der Unterrichtsszenarien anhand von Richtlinien aus einschlägiger Literatur sowie eigener Argumentation.

2 Programmieren in der Informationsgesellschaft

Die digitale Technologie ist in unterschiedlichen Ausprägungen immer mehr präsent in allen Aspekten unseres Alltags. Sowohl im privaten Bereich als auch in der Arbeitswelt sind wir umgeben von unterschiedlichsten Apparaten, deren Funktionalität durch den technologischen Fortschritt des digitalen Zeitalters ermöglicht wurde. Computer, Mobiltelefone, Autos, vermehrt auch Haushaltsgeräte und viele andere Utensilien sind mit Software ausgestattet, die oft ihren Kern bilden und erst dadurch den jeweiligen Funktionsumfang ermöglichen. Mittlerweile ist unser Alltag ohne diese Geräte unvorstellbar geworden.

Der Trend neigt dazu, dass digitale Technologie sowohl quantitativ als auch qualitativ in all unseren Lebensbereichen weiter ausgebaut wird. Die Technologieindustrie ist ständig am Wachsen und somit auch der Bedarf an geeigneten Fachkräften, die dieses Wachstum erhalten und vorantreiben können. Um dieser Tendenz gerecht zu werden, wird eine ausreichende Zahl an Personen benötigt, die sich für den Beruf als Softwareentwickler entscheiden und den entsprechenden Bildungsweg einschlagen.

Aber nicht nur für Fachpersonen sind Kenntnisse des Programmierens relevant. Viele Benutzer von Technologieerzeugnissen bevorzugen es, nicht lediglich deren passive Konsumenten zu sein, sondern kreativer Prosumer zu werden, welche die Technologie zwar verwenden, aber auch mit ihrer internen Funktionsweise vertraut sind und sie auf eigene Bedürfnisse anzupassen wissen. Des Weiteren werden beim Aneignen von Programmierkenntnissen in der Regel auch weitaus mehr Fähigkeiten als bloße Kommunikationsmethoden mit einem Rechner in Form von Syntax einer Programmiersprache vermittelt. Durch algorithmische Vorgänge und eine für den Novizen neue Denkweise, die zum Programmieren erforderlich ist, wird auch generell das analytische und strukturierte Denken, das Abstraktionsvermögen sowie die Logik gefördert. Diese Kompetenzen sind in der Programmierung sehr wichtig, können aber praktisch in allen Lebenssituationen zur Anwendung kommen.

Hinzu kommt, dass durch die steigende Technologisierung und ständige Interaktion mit entsprechenden Geräten im Alltag ein gewisser Grad an Hintergrundwissen und Verständnis der jeweiligen internen Vorgänge als selbstverständlich in der heutigen Informationsgesellschaft gesehen werden kann. Zumindest die Basiskenntnisse des Aufbaus einer Software und die Grundprinzipien der Programmierung werden sehr wahrscheinlich in absehbarer Zukunft zur Allgemeinbildung gehören.

Zusammenfassend sind wohl folgende Worte sehr passend, um die Bedeutung vom Programmieren im digitalen Zeitalter auszudrücken:

„*Everyone should learn how to program! Programming is not just a specialty discipline limited to computer science majors. Programming is a way of thinking that is useful to everyone.*“ (Van Roy, Armstrong, Flatt, & Magnusson, 2003, p. 270)

2.1 Die größten Hürden beim Erlernen vom Programmieren

Das Erlernen von Programmieren ist ein komplexer und vielschichtiger Prozess, der mit vielen Herausforderungen verbunden ist. Etliche Studien über Probleme, die dabei auftreten können, wurden

verfasst und bieten einen Überblick darüber, wie Novizen in der Programmierung zurechtkommen und auf was für Hürden sie dabei treffen. Unter anderem können folgende meist beobachtete und/oder von Anfängern am schwierigsten wahrgenommene Probleme hervorgehoben werden:

- Programmieranfänger haben generell einen Mangel an Fähigkeit algorithmisch zu denken. Sie tun sich schwer eine Problembeschreibung oder Aufgabenstellung zu analysieren, sie entsprechend modular in Teilaufgaben zu zerlegen und zu implementieren (McCracken et al., 2001).
- Es ist besonders schwierig, parallel mit algorithmischen Denkmustern und Problemlösungsansätzen, die notwendigerweise für die Programmierung bereits am Anfang vermittelt werden müssen, noch zusätzlich die Syntax einer Sprache zu bewältigen. Für Anfänger stellt das eine besonders große Hürde dar (Denny, Luxton-Reilly, Tempero, & Hendrickx, 2011).
- Aufbauend auf die Syntaxprobleme konnten auch generelle Schwierigkeiten beim Lesen und Verstehen von Codeausschnitten sowie beim Verständnis von grundlegenden Programmierkonzepten und Regeln festgestellt werden (Lister et al., 2004).
- Fehlermeldungen des Compilers bzw. Interpreters beim Ausführen eines Programms werden meistens als zu kurz und technisch empfunden. Es gibt keine visuelle Repräsentation, keine Fehlermeldung mit korrekten Beispielen etc. (Nienaltowski, Pedroni, & Meyer, 2008).
- Anfänger sind überfordert durch das gleichzeitige Erlernen vieler neuer Teilbereiche des Programmierens. Die Aneignung von algorithmischer Denkweise, problemlösenden Fähigkeiten, Bewältigung von Syntax und Semantik einer Programmiersprache, das Begreifen von Programmierkonzepten, Kombination von neuem und vorhandenem Wissen – all das erfolgt parallel im Rahmen der Vermittlung von Programmierkenntnissen, was zusätzlich an deren Komplexität beiträgt (Linn & Dalbey, 1989).
- Motivationsmangel durch nicht ansprechende Aufgabenstellungen, die nicht als Interesse weckend wahrgenommen werden (wie z.B. erstellen von Primzahlenlisten oder Sotieralgorithmen)

Alle oben genannten Probleme, mit denen Anfänger in der Programmierung konfrontiert werden, dürften deutlich demotivierend wirken und haben natürlich auch Einfluss auf die Entscheidung über die

Fortsetzung oder nächstmögliche Ausscheidung aus dem (freiwilligen) Unterricht, wofür sich durchschnittlich ca. 33% entscheiden (Watson & Li, 2014). Um dem entgegenzuwirken, sollte auf die Hürden eingegangen und ihnen soweit wie möglich mit entsprechenden Modalitäten im Lehrprozess entgegengewirkt werden.

Die von den Anfängern wahrgenommenen Schwierigkeiten werden jedoch auch dann wahrscheinlich nur gemildert werden und nie vollständig aus dem Weg geräumt sein. Deshalb gilt es, neben dem WIE – der Unterrichtsmethodik, auch an das WARUM heranzugehen und Ansätze mit möglichst motivierenden Aspekten der Programmierung für Novizen zu suchen. Durch entsprechende Motivation werden Hürden als Herausforderungen gesehen, für die es sich lohnt, auch die restlichen Schwierigkeiten mit Zuversicht anzugehen und sie zu bewältigen.

3 Didaktische Ansätze im Programmierunterricht (state of the art)

In den letzten Jahren sind vermehrt Versuche unternommen worden, Wege zu finden, die Einführung ins Programmieren möglichst einfach und motivierend zu gestalten. Abhängig von unterschiedlichen Voraussetzungen der jeweiligen Zielgruppe, wie z.B. dem Alter, Hintergrundwissen, möglichen Interessen etc., wurden verschiedene Ansätze erarbeitet.

Im folgenden werden einige der bekanntesten Ansätze für Einsteiger ins Programmieren kurz erläutert.

3.1 Spielerischer Ansatz

Im Rahmen dieses Ansatzes wird versucht, Grundzüge des Programmierens durch Spielerisches Auseinandersetzen mit Programmierkonzepten zu vermitteln. Als Vertreter könnten die graphische Programmierung und Mikrowelten erwähnt werden.

3.1.1 Graphische Programmierung

Die Graphische Programmierung wurde als Alternative konzipiert, um die abschreckenden Effekte

einer Codeansicht beim herkömmlichen Programmieren abzufangen. Des Weiteren wird der Fokus auf das Erlernen der wichtigsten Konzepte an sich gesetzt, ohne sich sehr auf die Syntax konzentrieren zu müssen. Diese wird vordefiniert und durch visuelle Elemente dargestellt, die meistens per drag and drop auf die „Programmieroberfläche“ übertragen und miteinander in Beziehung gebracht werden können. Syntaxfehler sind somit ausgeschlossen. Es wird praktisch visueller Pseudocode modelliert, der auch ausführbar ist.

Die graphische Programmierung erfolgt auf einer interaktiven Entwicklungsumgebung mit vielen multimedialen Elementen. Es können von einfachen Bewegungen simpler Animationen bis hin zu komplexen Spielen oder Geschichten modelliert und als Projekte für andere online zur Verfügung gestellt werden. Beispiele für solche graphische Entwicklungsumgebungen, die vor Allem zu Lernzwecken verwendet werden sind Scratch[1] (Abbildung 1), Blockly[2], Snap![3], Baltie[4] und andere.

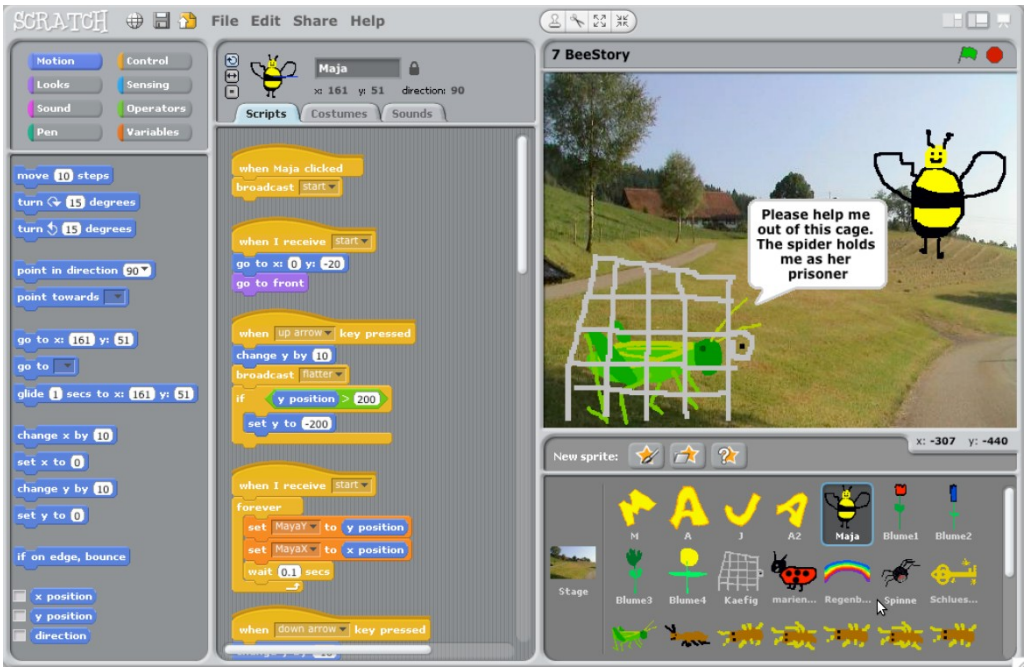


Abbildung 1: Szene aus Scratch. Quelle:(Resnick et al., 2009)

Die graphische Programmierung wird vor allem für Einsteiger als ein zu empfehlender Ansatz betrachtet, der Anfängern ihre Hemmungen und Unsicherheiten beim Umgang mit Rechnern und codierten Anweisungen durch spielerische und farbenreiche Elemente unterdrückt und trotzdem

1 https://scratch.mit.edu
2 https://developers.google.com/blockly
3 http://snap.berkeley.edu
4 https://www.sgpsys.com/en/whatisbaltie.asp

grundlegende Programmierkonzepte vermitteln kann (Meerbaum-Salant, Armoni, & Ben-Ari, 2013).

Der erste Kontakt mit einer Syntax wird möglichst einfach und intuitiv gestaltet, wodurch versucht wird, Programmieren zugänglich und ansprechend darzustellen. Zusammengefasst könnten folgende Elemente dieses Ansatzes festgehalten werden:

- Durch Visualisierung von Code werden Programmiergrundzüge anschaulich erlernt.
- Problemlösendes und algorithmisches Denken wird durch entsprechende Reihenfolge, Strukturierung, Bedingungen, Parametervergabe, etc. gefördert.
- Er eignet sich hauptsächlich für Spiele und Simulationen und ist kreativitätsfördernd.
- Er ist spielerisch-motivierend, was vor allem bei der jüngeren Population einen essenziellen Aspekt darstellt.

Die graphische Programmierung hat eine relativ niedrige Einstiegsschwelle, ist geeignet für erste Erfahrungen und ist schnell erlernbar. ScratchJr (Scratch Junior, eine Ableitung von Scratch)[5] eignet sich auch für Kinder ab 4 Jahren und mit Baltie können sogar kinder ab 3-4 Jahren lernen, ganz ohne jeglichen Text zu programmieren. Diese graphischen sprachen werden jedoch auch relativ schnell überwunden. Scratch wurde z.B. hauptsächlich für Kinder zwischen 8 und 16 Jahren entwickelt, wobei der Höhepunkt mit 12 erreicht wird (Resnick et al., 2009).

3.1.2 Mikrowelten

Andere Vertreter des spielerischen Ansatzes sind die sogenannten Mikrowelten. Das sind Lernentwicklungsumgebungen, die als eine Unterkategorie des grafischen Programmierens betrachtet werden können und beinhalten dementsprechend viele ähnliche Komponenten. Der zentrale unterschied stellt der Fokus von Mikrowelten dar: Es werden zur Vermittlung von Programmierkonzepten virtuelle Realitäten mit unterschiedlichen Akteuren und verschiedenen Situationen meistens in 3-D gestaltet, wodurch ganze Geschichten modelliert werden können.

Beispiele solcher Implementationen stellen zum Beispiel die Programmiersprachen Mama[6], StarLogo

5 http://www.scratchjr.org
6 http://www.eytam.com/mama

TNG[7] oder Alice[8] (Abbildung 2) dar, in denen per drag and drop Computer Animationen in 3-D erstellt werden können.

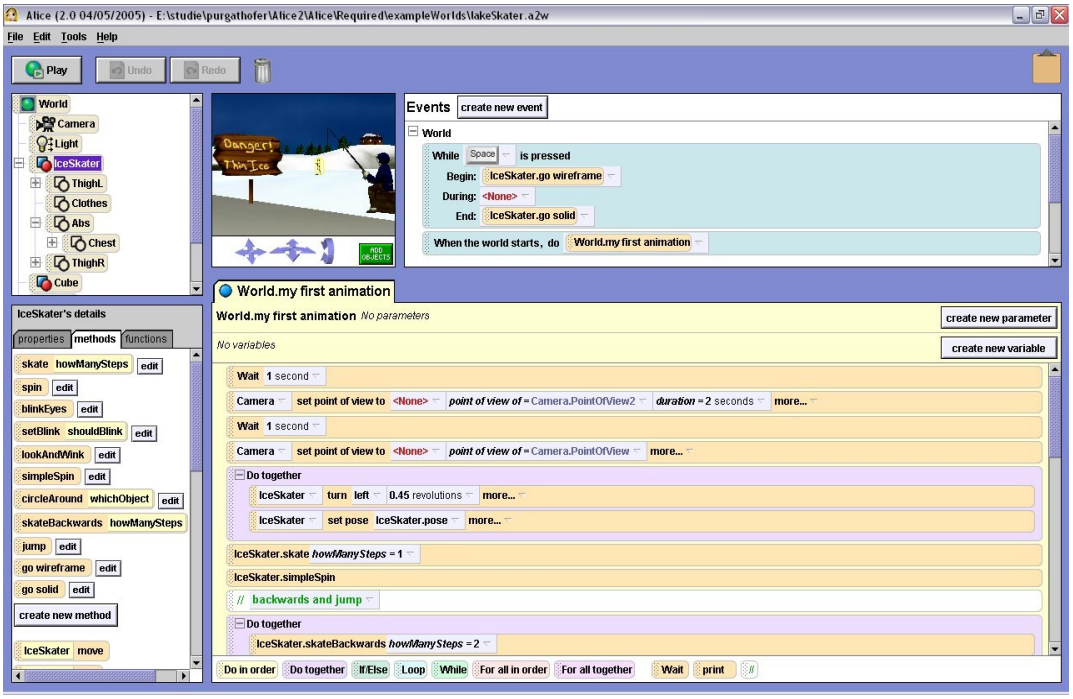


Abbildung 2: Szene aus Alice. Quelle: https://commons.wikimedia.org/wiki/File:Alice-2-screenshot.jpg

Storytelling Alice zum Erlernen von Programmierkonzepten, die Kelleher und Pausch basierend auf der Alice Programmiersprache in Ihrer Studie erforscht haben, zeigte auch eine sehr motivierende Wirkung beim Programmieren (Kelleher & Pausch, 2007). Insbesondere wird argumentiert, dass objektorientierte Konzepte wie Klassen und Methoden durch Geschichten und deren Akteure auf natürliche, intuitive Weise näher gebracht werden können.

Wie bei allen visuell-spielerischen Programmiersprachen sind auch Mikrowelten als Lernhilfe mit dem Fokus vor allem auf Kindern und Jugendlichen mit niedriger Einstiegsschwelle gedacht. Obwohl sie Spaß am Programmieren bewirken und die Kreativität fördern, stoßen ihre Einsatzmöglichkeiten relativ bald auf Grenzen, wenn der Rahmen der Programmierung außerhalb der Edukation erweitert werden soll.

7 http://education.mit.edu/portfolio_page/starlogo-tng
8 http://www.alice.org

3.2 Modellierungsansatz

Als eine Herangehensweise für etwas anspruchsvollere Programmiereinsteiger könnte etwa der Modellierungsansatz genannt werden. Er beruht hauptsächlich auf dem 'objects first' Paradigma, wobei der Fokus nicht auf dem Erstellen von farbenreichen Spielen und Geschichten ist, sondern auf dem Vermitteln von Programmierkonzepten durch schematisierte Darstellungen von Code auf einer höheren Abstraktionsebene. Der Code kann visuell durch Klassendiagramme modelliert werden, wobei die Einsicht in die zugrundeliegende Syntax der jeweiligen Programmiersprache ebenfalls möglich ist. Im Rahmen von diesem Ansatz könnte etwa der Programmiereinstieg durch den Java-Editor oder BlueJ erwähnt werden.

3.2.1 BlueJ

Eine Java - Entwicklungsumgebung, die seit 1999 für Ausbildungszwecke verwendet wird. Sie wurde gezielt dafür entwickelt, Novizen in das OOP einzuführen. Für diese besondere Eignung wird unter anderem die Einfachheit der Umgebung und Reduktion auf wesentliche Optionen hervorgehoben, um den Komplexitätsgrad des Tools in Grenzen zu halten. Bereits nach kürzester Zeit kann die Umgebung erlernt und zum Programmieren an sich gewechselt werden (Barnes & Kölling, 2009).

Das Zentrale Feature von BlueJ ist die visuelle Darlegung von Code durch ein UML-Klassendiagramm, das die Struktur der Applikation veranschaulichen soll. Es werden Klassen, Attribute, Methoden und die Beziehungen der Klassen zueinander dargestellt und Benutzer können per Mausklick mit den Elementen interagieren (Abbildung 3). Instanzen von Klassen können auf diese Weise einfach erstellt und deren Methoden anschließend aus dem Kontextmenü aufgerufen und Parameter in dafür vorgesehene Felder eingegeben werden, ohne eine Testklasse mit der main Methode erstellt zu haben. Durch Doppelklick auf Klassen kann der zugrundeliegende Java-Code im eingebauten Editor angezeigt und angepasst werden.

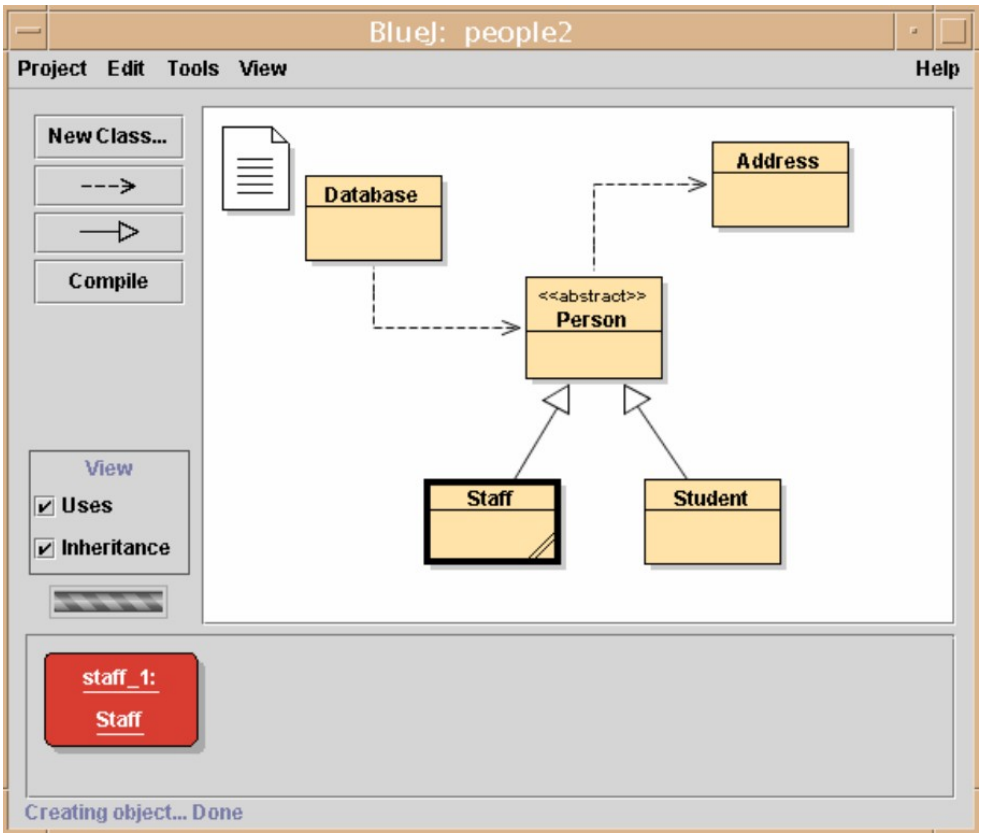


Abbildung 3: Beispiel der Modellierung in BlueJ. Quelle: (Kölling, Quig, Patterson, & Rosenberg, 2003)

BlueJ baut gänzlich auf dem 'objects first' Paradigma auf und konzentriert sich auf die Vermittlung dieser Konzepte. Daher wird als eines der wichtigsten Vorteile die Visualisierung hervorgehoben, dass Programme eine Sammlung von zusammenwirkenden Klassen sind und eine Struktur haben. Des Weiteren wird argumentiert, dass durch BlueJ der für Anfänger in der Regel schwer verständliche Unterschied zwischen Klassen und Instanzen leichter begreiflich gemacht werden kann, da ersichtlich ist, dass Klassen nur als Schablonen dienen, aus denen per Mausklick Objekte erstellt werden. Auch das Ändern von Objektzuständen und die Interaktion zwischen unterschiedlichen Objekten durch die Verwendung von Methodenaufrufen wird von Anfang an auf einfache Weise vermittelt (Kölling et al., 2003).

Die Lernumgebung weist somit ersichtliche Vorteile beim Erlernen der objektorientierten Konzepte auf, mit der Visualisierung als wesentlichem Bestandteil von BlueJ. Da diese sich jedoch in erster Linie auf die Struktur des Programms und dessen abstrakte Elemente begrenzt, dürften die optischen Faktoren, zumindest den Anreiz zur Programmierung betreffend, zu wünschen übrig lassen.

3.3 Entwicklung in professionellen IDEs

Manche Studien schlagen auch vor, die Einführung ins Programmieren gleich zu Beginn auf professionellen Entwicklungsumgebungen vorzunehmen, was natürlich für bestimmte Zielgruppen wie z.B. Kinder nicht anwendbar ist. Chen und Marx (2005).schlagen beispielsweise Eclipse[9] für den Programmiereinstieg vor.

Eclipse ist eine Entwicklungsumgebung, die in erster Linie für die Programmierung in Java am geeignetsten ist, aber durch viele Plug-in Möglichkeiten und Erweiterungen auch andere Sprachen, wie C#, C++, Python etc. unterstützt (Abbildung 4). Im folgenden werden die wesentlichen Vorteile der Verwendung von Eclipse beim Programmieren angeführt:

- Bei der Benutzung der IDE entstehen keine Kosten, da open source;
- Verfügbarkeit von Wizards für die benutzerfreundliche Erstellung von Klassen, Interfaces, Konstruktoren etc.;
- Das Schreiben/Lesen von Code wird durch Hervorhebung der Syntax vereinfacht, Fehler können leichter identifiziert und behoben werden;
- Keyword-Syntax, bereits definierte Variablen, sowie die Codestruktur (klammern, Semikolon, etc.) werden während der Codierung automatisch vervollständigt, was das Schreiben von Programmen deutlich beschleunigt;
- Auf potenzielle Fehlerquellen wie nicht geschlossene Klammern, nicht deklarierte Variablen etc. wird während des Codierens durch Kompilierung in Echtzeit farblich hingewiesen;
- Nützliche Hinweise werden während der Codierung angegeben (z.B. Parametertyp/-anzahl bei der Verwendung von vorhandenen Methoden);
- Erweiterbarkeit auf andere Sprachen durch Plugins, wobei dasselbe look and feel in der bereits vertrauten Entwicklungsumgebung bleibt;
- Programmieren gleich am höchsten Level: Eclipse ist eine professionelle IDE, die auch in der Industrie verwendet wird, was als zusätzlicher Anreiz gesehen werden und das Selbstvertrauen stärken kann.

9 https://eclipse.org

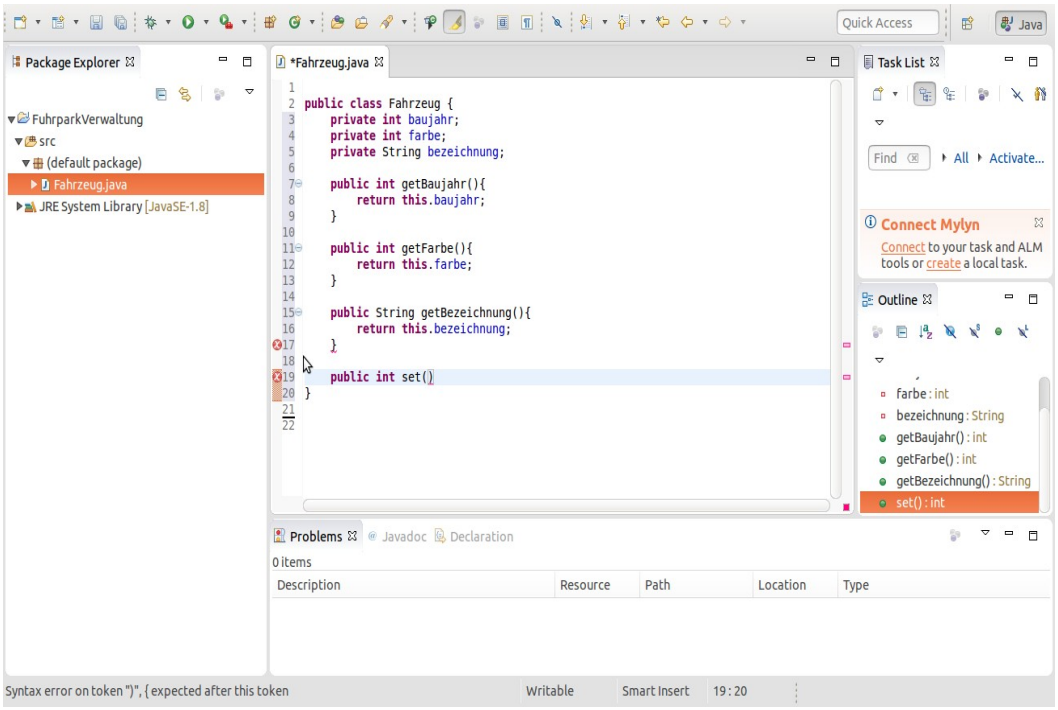


Abbildung 4: Eclipse Entwicklungsumbegung im Einsatz (eigene Darstellung)

Auch wenn eine Entwicklungsumgebung wie Eclipse eine erhebliche Unterstützung bei der Erstellung von Software darstellt, hat sie vor allem im Zusammenhang mit Programmieranfängern auch ihre Nachteile, wovon zu den wichtigsten folgende gehören:

- Komplexität: eine professionelle IDE ist ein mächtiges Tool, das jedoch für Anfänger einen Überfluss an Optionen enthält und das gesamte Unternehmen noch komplizierter erscheinen lässt;
- IDEs sind überwältigend für Novizen, die sich gleichzeitig mit algorithmischem Denken und der grundlegenden Syntax von Java auseinandersetzen müssen;
- Ohne Anweisungen seitens des Lehrpersonals und die anfangs kontinuierliche Unterstützung ist die Entwicklungsumgebung nicht einfach zu bedienen;
- Es benötigt eine gewisse Einarbeitungszeit, sich an die Arbeit in der IDE zu gewöhnen, der Prozess weist eine steile Lernkurve auf;
- Fehlende Visualisierung: Mangel an entsprechender Veranschaulichung vom geschriebenen Code;

- Unerwünschte Fokusverlagerung: die Syntaxhervorhebung, Autovervollständigung, Wizards etc. erleichtern zwar das Schreiben von Code, steuern aber wenig zum wesentlichen konzeptuellen Verständnis des Programmierens bei.

Um den Nachteilen einigermaßen entgegenzuwirken, wurden Plugins für pädagogische Zwecke wie GILD (Storey et al., 2003) und Penumbra (Mueller & Hosking, 2003) entwickelt. Mit denen wird Eclipse für Lernzwecke angepasst, viele unnötige Eclipse-Optionen werden abstrahiert und nur die dafür wesentlichen Ansichten und Optionen dargestellt.

Dennoch wird vorgeschlagen, mit Eclipse zwar sehr früh zu beginnen (zweite Hälfte des Semesters), aber nicht direkt mit der IDE ins Programmieren einzusteigen. Die Einführung sollte mit dem JDK gemacht werden, damit generelle Programmierkonzepte und Struktur beigebracht werden und die Entwicklungsumgebung wirklich nur als Tool verstanden wird, mit dem höhere Produktivität erzielt werden kann. Des Weiteren wird hervorgehoben, dass nicht die Komplexität der IDE die größte Schwierigkeit beim Programmieren darstellt sondern der Mangel an problemlösendem Denken und algorithmischer Strukturierung von Aufgabenstellungen (Chen & Marx, 2005).

4 Motivation

Im vorigen Kapitel wurden mehrere didaktische Ansätze und Hilfsmittel für die Einführung in das Programmieren vorgestellt, jedoch ist wichtig zu erwähnen, dass jede Herangehensweise sowohl ihre Vor- als auch Nachteile hat. Nicht eine ist universell und entspricht allen pädagogischen Anforderungen, sondern es kommen abhängig von der Zielsetzung und der Zielgruppe unterschiedliche Ansätze zur Anwendung. Vielmehr wird empfohlen, in einer sich ständig ändernden und weiterentwickelnden Disziplin wie der Informatik kontinuierlich nach neuen Strategien der Wissensvermittlung zu suchen und diese in der Praxis zu bewähren (Curricula, 2001). Für einen erfolgreichen Unterricht sind didaktische Methoden und Werkzeuge jedoch nur ein Teil der erforderlichen Maßnahmen. Eine zentrale Rolle bei Lernprozessen, so auch beim Erlernen von Programmierkonzepten, nimmt die Motivation ein, weshalb sie möglichst gefördert werden sollte.

„Motivation wird als Sammelbezeichnung für eine Vielzahl von Prozessen konzipiert, deren gemeinsamer Kern darin besteht, dass ein Individuum sein Verhalten um der gewünschten Folgen willen auswählt und hinsichtlich Richtung und Aufwand steuert“ (Schmalt & Heckhausen, 1992 zitiert nach Deimann, 2002, S.63).

Bezogen auf das Lernen kann Motivation als ein Bestreben betrachtet werden, sich bestimmtes Wissen oder Fähigkeiten anzueignen. Deimann (2002) befasst sich intensiv mit dem Begriff der Motivation unter Bezugnahme auf andere einschlägige Studien. Obwohl dieses Forschungsgebiet sehr breit ist und viele unterschiedliche Theorien und Sichtweisen beinhaltet, können bestimmte generelle Merkmale hervorgehoben werden. Unter anderem wird dargelegt, wie Motivation durch eine Korrelation von den Eigenschaften einer Person bzw. deren Interessen und den Anreizen, die eine bestimmte Situation dementsprechend bieten kann, entsteht. Grundsätzlich werden zwei Ausprägungen der Motivation unterschieden: die extrinsische und intrinsische. Extrinsische Motivation ist gegeben wenn eine Person mit dem Ziel handelt, nur ein bestimmtes Ergebnis zu erzielen und keinen persönlichen Bezug zu der jeweiligen Handlung hat. In der Regel geht es dabei um die Erwirkung von Belohnungen (z.B. Lernen nur um die Prüfung zu bestehen oder Arbeiten nur um ein Gehalt zu bekommen) oder um Meidung von Strafen (z.B. pünktlich zur Arbeit erscheinen um nicht entlassen zu werden). Bei der intrinsischen Motivation hingegen spielt der innere Bezug zur Handlung eine wesentliche Rolle, wobei externe Faktoren von sekundärer Bedeutung oder gar nicht gegeben sind. Beispielsweise könnte in etwa das freiwillige Aneignen von Wissen angeführt werden, das für keine Prüfung benötigt wird und für das man auch keine materielle Belohnung erhält. Wenn sich z.B. jemand mit dem Fach Physik auseinandersetzt, nur um die Gesetzmäßigkeiten und Phänomene der Welt zu verstehen und Freude an dem Wissen an sich und seiner eigenen Kompetenz in diesem Fach empfindet, dann ist seine Handlung intrinsisch motiviert.

Als Vorteile der intrinsischen Motivation gelten ihre Nachhaltigkeit und Selbsterhaltung und da sie jenseits von Belohnung und Strafe wirkt und mit dem eigenen Interesse verbunden ist, resultiert sie üblicherweise im tieferen und zeitlich intensiveren Auseinandersetzen mit dem jeweiligen Thema. Dies hat tiefgründiges und umfangreiches Wissen zur Folge, weswegen bevorzugt diese Art der Motivation bei den Lernenden angeregt oder gefördert werden sollte.

4.1 Motivation und neue Technologien

Eine der Möglichkeiten, den Motivationsgrad beim Lernen zu steigern, wäre die Verwendung von neuen Medien und Technologien, wie unter anderem Deiman (2002) in seiner Studie argumentiert. Es wurde gezeigt, dass die Einbindung eines neuen Mediums in den Lernprozess das Interesse bei den Lernenden weckt, was als generelle Regel auch auf Smartphones übertragbar ist. Viele Studien haben sich in den letzten Jahren mit der Verwendung von mobilen Technologien, insbesondere Smartphones, beschäftigt und konnten sowohl bei deren bloßen Benutzung positive Effekte und gesteigerte Motivation zu deren weiteren Gebrauch finden (Kim, Kim, & Wachter, 2013), als auch beim Einsatz im Bildungsbereich (Ciampa, 2014; Holz, Leonhardt, & Schroeder, 2011; Joosten, 2010; Schwabe & Göth, 2005; Su & Cheng, 2015; Yamamoto & Wakahara, 2013).

Eines der wichtigsten Merkmale für die motivationale Förderung beim Lernprozess stellt die Interaktivität dar. Als interaktiv werden die Vorgänge verstanden, durch die der Benutzer mit seinen Aktivitäten in Prozesse eingreifen und sie in gewisser Weise steuern kann, wobei die Auswirkungen direkt sichtbar sind. Interaktivität bzw. deren Umsetzung bei Lernprozessen ist essenziell für die Erhaltung der Lernmotivation, denn Rückmeldungen vom Lernsystem fördern eine aktive und strebsame Wissenserweiterung (Deimann, 2002).

Interaktivität ist durch den Ansatz, durch App-Entwicklung Programmieren zu lernen, durchaus gegeben, da Apps notwendigerweise mit interaktiven GUIs verbunden sind.

Im Rahmen der Motivationsforschung gibt es eine Vielfalt von Ansätzen, die Empfehlungen geben, wie der Lernprozess zu gestalten ist, damit wirksame Effekte erzielt werden. Eines der herausragenden ist das ARCS-Modell (Tabelle 1) von Keller (1983). Der Name Steht für die Anfangsbuchstaben der enthaltenen Komponenten, die wesentlich für die Motivation sind.

Tabelle 1: Komponenten des ARCS-Modells – Wissen motivierend vermitteln (angepasst nach Keller, 1983)

Initial	Beschreibung	Erläuterung
A	Attention	Aufmerksamkeit bzw. das Interesse des Lerners erlangen und aufrechterhalten; erster Schritt jeder Lernmotivierung.
R	Relevance	Relevanz, Bedeutsamkeit des Lehrstoffs vermitteln
C	Confidence	Positive Erfolgserwartung geben
S	Satisfaction	Befriedigung, Zufriedenheit bieten

Im Rahmen des zentralen Themas, mit dem sich diese Arbeit befasst, könnte das ARCS-Modell in der App-Programmierung folgende Ausprägung finden(Tabelle 2):

Tabelle 2: Anwendung des ARCS-Modells in der App-Programmierung (adaptiert nach Keller, 1983)

Initial	Beschreibung	Anwendung in der Einführung durch App-Programmierung
A	Attention	Interesse durch Umgang mit neuer Technologie geweckt. Smartphones ziehen die Aufmerksamkeit auf sich und verleiten das Individuum so zur längeren und weiteren Beschäftigung
R	Relevance	Die Programmierkonzepte werden mit realitätsnaher Anwendung vermittelt. Es werden Fähigkeiten beigebracht, mit denen zeitnah nützliche Apps erstellt werden können.
C	Confidence	Es wird von Vorhinein in Aussicht gestellt, dass der Lernende befähigt wird recht zeitnah programmieren zu lernen und eigene Apps zu entwickeln.
S	Satisfaction	Sowohl das erlernte know-how als auch ein veranschauliches Fertigprodukt in Form (zumindest) eines eigenständig erstellten Apps ist das Resultat, welches Zufriedenheit verschaffen soll.

Da das Modell sehr prominent und auch empirisch gestützt ist, erscheint dessen Verwendung auch bei der Einführung ins Programmieren durch App-Entwicklung vielversprechend.

5 Smartphone App Entwicklung

In den vorigen Kapiteln wurden Erkenntnisse erläutert, dass die Vermittlung von Wissen in einer für die jeweilige Zielgruppe didaktisch aufbereitete Form geschehen soll und dabei auch die Motivation der Lernenden eine sehr wichtige Rolle einnimmt. Die Programmierung ist dabei keine Ausnahme. Diese Arbeit schlägt jedoch einen anderen Ansatz für die Einführung ins Programmieren vor, als die in Kapitel 3 vorgestellten, und zwar basierend auf den in Kapitel 4 dargelegten motivationsfördernden Effekten der Smartphoneverwendung. (Generell dürfte die Interaktion mit GUIs, die ein wesentlicher Bestandteil von Apps sind, attraktiver gelten als Ein- und Ausgaben in einer Konsole, was der traditionelle Weg für die Programmiereinführung ist.)

An dieser Stelle sollte erwähnt werden, dass es bereits teilweise ähnliche Vorstöße gibt, Programmierkenntnisse unter Verwendung von Smartphones zu vermitteln. Catrobat[10] und AIDE (Android IDE)[11] zählen unter anderem zu deren Vertretern.

Catrobat ist eine Scratch-ähnliche graphische Entwicklungsumgebung für mobile Geräte (Slany, 2012) und AIDE ist eine App, mit der direkt am Smartphone in Java programmiert werden kann ("AIDE - Android IDE," 2015). Der Nachteil dabei ist die relativ kleine Bildschirmfläche und die schwierig zu gestaltende Codeeingabe für die Erstellung von Programmen über die Smartphone-Tastatur.

Das Ziel dieser Arbeit ist jedoch, im Gegensatz zu Catrobat, Programmieren in einer höheren, nicht-graphischen Sprache, die auch in der Industrie verwendet wird, durch App-Entwicklung beizubringen. Als Abgrenzung zu AIDE soll keine Beschränkung auf Java gegeben sein und die Programmierkenntnisse sollen durch Entwicklung von Apps, jedoch nicht in Apps vermittelt werden.

Der in dieser Arbeit verfolgte Ansatz setzt eine höhere Einstiegsschwelle der Lernenden voraus und eignet sich eher für Jugendliche und Erwachsene, die entweder keine Vorkenntnisse haben oder womöglich erste Erfahrungen mit der allgemeinen Programmierlogik gesammelt haben (z.B. in Scratch) und nun auf einem höheren Level programmieren möchten.

10 http://www.catrobat.org
11 http://www.android-ide.com

5.1 App Grundlagen

Als Applikation oder abgekürzt App kann im weiteren Sinne jegliche Anwendungssoftware verstanden werden, die auf einem Betriebssystem aufbaut und bestimmte Funktionalität als Erweiterung anbietet. Im engeren Sinne, durch den heutzutage weit verbreiteten Sprachgebrauch bedingt, werden als Apps Anwendungen für mobile Geräten wie Smartphones und Tablets bezeichnet, was auch in dieser Arbeit als Begriffseingrenzung herangezogen wird.

Auch in dem engeren Sinne ist der Begriff sehr weit angelegt und kann eine weite Bandbreite von möglichen Ausprägungen in Umfang und Komplexität der Anwendung haben – von einfachen Taschenrechnern bis hin zu Apps, an deren Entwicklung ganze Teams oder Unternehmen tätig sind. Apps können vorinstalliert auf den jeweiligen Geräten sein oder können in der Regel von online Plattformen wie Google Play und App Store betriebssystemspezifisch heruntergeladen und installiert werden. Manche werden kostenlos, andere hingegen kostenpflichtig angeboten. Es handelt sich meistens um relativ kleine Anwendungen in Form von einfachen Werkzeugen wie Terminkalender, Währungskonverter, Stoppuhr, etc., aber auch um Programme wie Internet Browser, Media-Player, oder auch Spiele. In letzter Zeit hat die Anzahl der App-User, gemessen an der Anzahl von angebotenen Apps und deren Downloads drastisch zugenommen, mit steigender Tendenz. Beispielsweise hat hat die Plattform App Store von Apple im Jahr 2014 ungefähr 1.2 Mio. Apps angeboten und eine Downloadanzahl von 75 Mrd. vermerkt. Im Jahr zuvor waren es 900.000 Apps mit 50 Mrd. Downloads. Der Konkurrent, Google Play, konnte ebenfalls mit ca. 1.2 Mio. Apps ähnliche Zahlen aufweisen (Perez, 2014).

Diese Angaben weisen deutlich darauf hin, dass die App-Benutzung weit verbreitet und sehr beliebt ist. Die Information sollte zunutze gemacht werden, um diese gegenwärtig aktuelle Technologie als Motivationsfaktor in den Unterricht einzubeziehen.

5.2 Technische Rahmenbedingungen

Für die Entwicklung von Apps für mobile Geräte, werden spezielle Frameworks mit entsprechenden Libraries benötigt. In der Regel findet die Programmierung in Entwicklungsumgebungen statt, die entweder auf Apps spezialisiert sind (z.B. Android Studio von Google für Android basierte Apps[12],

12 https://developer.android.com

Xcode für iOS Betriebssysteme[13]) oder in generischen IDEs, wie Eclipse, entsprechende Erweiterungen für App-Programmierung beinhalten. Für einen effizienteren Softwareentwicklungsprozess wird der Code üblicherweise zunächst auf dem lokalen Rechner unter Verwendung von Emulatoren getestet. Diese dienen zur virtuellen Abbildung von Konfigurationen und Verhalten/Funktionalitäten von mobilen Geräten wie Smartphones, ohne dass diese dem Entwickler in Wirklichkeit zur Verfügung stehen müssen. Dadurch können Kosten für das Testen von Applikationen auf unterschiedlichen Geräten vermieden werden und die Entwicklungszeit wird deutlich verringert durch Umgehen von Kompilierung, Transfer und Installation am mobilen Gerät jeglicher kleinen Codeänderung in der Testphase.

In der vorliegenden Arbeit wurde Android als das Ziel-Betriebssystem zum Testen der Apps verwendet., wobei das Gesamtkonzept nicht auf Android beschränkt ist.

Im Folgenden werden weitere Rahmenbedingungen erläutert, die für diese Arbeit herangezogen wurden.

5.2.1 Die Frage des Paradigmas

Bezüglich der Gestaltung von Lehrprozessen für die Einführung in das Programmieren gibt es in der einschlägigen Literatur etliche Studien mit unterschiedlichsten Empfehlungen. Die Fragen *was* und insbesondere *wie* Unterrichtet werden soll sind bekannte Kontroversen unter Experten dieses Fachbereichs. Eines der zentralen Streitpunkte ist das Programmierparadigma und zwar welches im Unterricht für Novizen herangezogen werden sollte. Darunter fallen die zwei am meisten diskutierten – das prozedurale und das objektorientierte.

Unter dem prozeduralen Paradigma wird der Ansatz verstanden, Programme grundsätzlich als aufeinander folgende Anweisungen zu schreiben. Diese können auch in kleinere Unterprogramme oder sog. Prozeduren zerlegt werden, die für bestimmte Aufgaben im Rahmen des Gesamtprogramms verantwortlich sind. Sie können bei Bedarf meistens mit Parameterübergabe aufgerufen werden und führen dabei die entsprechende Teilaufgabe durch und/oder liefern ein Ergebnis zurück, die für den weiteren Ablauf im Code notwendig ist. Die wesentlichen Vorteile dieser Strukturierung von Code sind bessere Übersichtlichkeit und Vermeidung von redundanten Codeausschnitten.

13 https://developer.apple.com/xcode

Das objektorientierte Paradigma hingegen versucht Programme nicht als eine bloße Folge von Anweisungen darzustellen sondern versucht sie, der Wirklichkeit entsprechend, als Sammlung von zusammengehörenden Codemodulen – Objekten – zu strukturieren und miteinander interagieren zu lassen, um bestimmte Aufgaben zu erledigen. In der Realität kann alles greifbare als ein Objekt identifiziert werden, das entweder einfach oder aus anderen Objekten zusammengesetzt ist, bestimme Eigenschaften hat und Funktionen erfüllt. Diese Konzepte werden im objektorientierten Paradigma verwendet, um den Code entsprechend zu strukturieren. Häufig angeführte Vorteile dieses Vorgehens sind die bessere intuitive Verständlichkeit von auf diese Weise abgebildetem Code, Modularität, leichtere Wiederverwendbarkeit von Code, einfachere Wartbarkeit, etc.

Beide Paradigmen, die hier in aller Kürze erläutert wurden, haben ihre Befürworter, die sich bemühen, durch verschiedene Studien ihre Ansicht zu belegen. Unter anderem wird hervorgehoben, dass es für Programmieranfänger eine Herausforderung ist, sich im Rahmen der objektorientierten Programmierung viele Konzepte auf einmal aneignen zu müssen, da bereits das einfachste Objekt, dessen Instanziierung und Aufruf viele Konzepte wie Variablen, Attribute, Methoden, Parameter etc. beinhaltet.

Auch wenn in der Fachliteratur jedoch der Eindruck entsteht, dass sich in letzter Zeit das OO-first Paradigma durchzusetzen vermag (Davies, Polack-Wahl, & Anewalt, 2011), und die Argumente, die dafür sprechen, die Oberhand gewinnen, ist dieser Umstand nicht der entscheidende Faktor bei der Auswahl dieses Paradigmas für die vorliegende Arbeit gewesen. Vielmehr sollte die Motivation, die durch die Verwendung von Smartphones als aktueller Technologie gefördert werden soll, die Fokussierung auf ein bestimmtes Paradigma in den Hintergrund stellen. Der Einstieg ins Programmieren durch objektorientierte Konstrukte ist nicht unbedingt durch Befürwortung von Objektorientiertheit entstanden, sondern eher durch dem Umstand bestimmt worden, dass Apps grundsätzlich nur durch Verwendung von vorhandenen Klassen und deren Erweiterung programmiert werden können.

5.2.2 Auswahl einer Programmiersprache

Ein weiterer Streitpunkt unter den Experten ist die Frage, welche Programmiersprache im Unterricht für Anfänger verwendet werden soll. Auch hierfür gibt es unzählige Studien, die Argumente für ihre eigene, teils völlig entgegengesetzte, Standpunkte erbringen. Ohne tiefer auf die Diskussionen

einzugehen, wird nur oberflächlich der für diese Arbeit relevante Diskurs erwähnt und eine entsprechende Empfehlung aufgrund von ausgewählten Kriterien gegeben.

Durch den gegebenen Umstand, dass Apps in Form von Ableitungen von Klassen entwickelt werden, und dass die Objektorientiertheit schon von Anfang an herangezogen werden muss, wurde die Auswahl einer geeigneten Programmiersprache schon zu Beginn ziemlich eingeschränkt. Es musste sich um eine Sprache handeln, die entweder zur Gänze objektorientiert ist, oder objektorientiertes Programmieren unterstützt.

Unter weiteren wichtigsten Kriterien für die Auswahl einer Sprache, die womöglich aus technischer Sicht nicht unerlässlich, jedoch im Allgemeinen durchaus von Bedeutung gewesen sind, waren folgende:

- App-Unterstützung – die Programmiersprache muss Frameworks und Libraries anbieten, die eine App-Entwicklung überhaupt ermöglichen.
- Simplizität und Mächtigkeit – die Syntax der jeweiligen Programmiersprache sollte möglichst einfach, jedoch mächtig genug sein, über den Unterricht hinaus Anwendung finden zu können. Wenn eine Wahl zwischen Mächtigkeit und Simplizität zu treffen ist, so ist die Simplizität zu bevorzugen, da bei der Einführung ins Programmieren die generellen Konzepte vermittelt werden sollen und es nicht um die Ausschöpfung der Möglichkeiten einer Sprache geht.
- Verbreitung – da die Lernprozesse heutzutage nicht auf den Unterricht beschränkt sind, sollte die Sprache möglichst weit verbreitet sein, mit vielen frei zugänglichen Lernmaterialien, vorhandenen Libraries, einer etablierten online community und Foren.
- Relevanz – die Programmiersprache sollte vorzugsweise nicht nur für Bildungszwecke tauglich sein, sondern in der Industrie verwendet werden und möglichst dem globalen Trend folgen. Somit könnte die Notwendigkeit, relativ bald die Syntax einer neuen, anwendbaren Programmiersprache erlernen zu müssen, vermieden werden. Des Weiteren ist eine Industrie-relevante Sprache ein Motivationsfaktor, der nicht zu vernachlässigen ist.
- Der Einsatz in der App-Entwicklung sollte möglichst unkompliziert sein – die Vorgehensweise Apps zu erstellen, die dafür notwendigen Libraries und in Betracht kommenden Entwicklungsumgebungen dafür sollten für Anfängern relativ einfach bedienbar sein.

Unter Berücksichtigung der genannten Kriterien und der im Vorhinein erwähnten Rahmenbedingungen sind nach einer Recherche die Sprachen Java und Python in die engere Wahl gekommen.

Java ist die wohl am weitesten verbreitete Sprache, die auch in der Industrie für professionelle Softwareentwicklung verwendet wird. Mit Java programmiert man sozusagen gleich am höchsten Niveau. Das ist auch die Standardsprache, die im Android Studio, der offiziellen Entwicklungsumgebung von Google für Android Apps herangezogen wird.

Java ist zwar sehr mächtig, hat jedoch ihre Nachteile hinsichtlich ihrer Komplexität in Syntax und Semantik,.was sich vor allem bei Programmieranfängern am deutlichsten auswirkt.

Viele Studien besagen, dass Python gerade für Novizen Vorteile bietet, welche die generellen Programmierkonzepte zu verstehen versuchen und sich nicht an die Grenzen der Möglichkeiten einer Programmiersprache begeben (Goldwasser & Letscher, 2008; Kasurinen & Nikula, 2007; Patterson-McNeill, 2006; Radenski, 2006).

Unter anderem wird betont, dass in Python eine einfachere Syntax verwendet wird. Die oft zur Verwirrung führende Klammersetzung wird durch verpflichtende Einrückungen ersetzt, was implizit auch für eine bessere Strukturierung von Code zwecks Übersichtlichkeit beiträgt. Des Weiteren müssen beispielsweise Datentypen von Variablen und Methoden nicht explizit deklariert werden.

In einer quantitativen Evaluation von unterschiedlichen Ansätzen für die Einführung in die Programmierung wurde gezeigt, dass die Verwendung einer syntaktisch einfachen Sprache wie Python im Gegensatz zu komplexeren wie Java einen fördernden Effekt auf das Erlernen von Programmierkonzepten hat (Koulouri, Lauria, & Macredie, 2014).

Der wichtigste Grund, Python für die Einführung ins Programmieren zu wählen, wie auch Rufinus und Kortsarts (2006) argumentieren, ist, dass es eine mächtige, jedoch syntaktisch simple Sprache ist, welche den Unterrichtsfokus von komplexen syntaktischen Strukturen auf problemlösendes Denken und Programmierkonzepte verlagern lässt.

Auch wenn womöglich andere Studien zu unterschiedlichen Ergebnissen hinsichtlich der Empfehlung einer ersten Programmiersprache gekommen sind, wurde für diese Arbeit Python ausgewählt. Neben den erwähnten Argumenten waren folgendes die ausschlaggebenden Punkte dafür:

- Unkomplizierte Anwendung bei Programmierung von Apps: Zu Python wird eine Library heruntergeladen, welche die benötigten Klassen für Apps beinhaltet. Es ist keine Entwicklungsumgebung notwendig, die zusätzlichen Overhead beim Lernen verursachen würde. Das Erstellen und Testen von Code kann im einfachen Texteditor bzw. durch Konsolenaufruf (oder Doppelklick auf Datei, nach entsprechender Einrichtung) erfolgen.
- Performanz beim Testen von Code: vor Allem bei Anfängern sind sehr kleine Schritte beim Programmieren mit vielen Ausführungen von Code und Tests zu erwarten. Daher ist die Performanz eines Smartphone-Emulators auf dem Rechner von wesentlicher Bedeutung, da lange Zeiten zum Ausführen von Code für den Unterricht nicht zumutbar wären. Im Rahmen dieser Arbeit wurde der Emulator vom Java-basierten Android Studio auf einem Rechner mit 4 GB RAM und einer Intel quad-core CPU mit 2.4 GHz getestet. Jeder Aufruf nach jeglicher Codeänderung dauerte ca. 40 Sekunden mit erheblicher RAM- und CPU Belastung. Python hingegen bietet einen Codeaufruf von Apps in der Konsole, der sie praktisch in Echtzeit als bedienbare Fenster am Rechner darstellt.
- Die in Python entwickelten Apps sind plattformübergreifend und können auf den meist verbreiteten Betriebssystemen – Android, iOS, Windows – ausgeführt werden.

5.2.3 Werkzeuge

Nach Eingrenzung der Rahmenbedingungen werden nun kurz die Tools erläutert, die für die App-Programmierung verwendet werden.

An erster Stelle ist natürlich die Python Umgebung samt Interpreter als notwendige Voraussetzung zu erwähnen. Darauf aufbauend wird Kivy[14] benötigt – eine Python Klassenbibliothek für die Entwicklung von Apps für mobile Geräte sowie andere Multi-Touch Anwendungsprogramme. Diese steht frei zum Download und kann sowohl in eine vorhandene Python Installation eingebunden oder auch als standalone tool ohne Installation verwendet werden.

Kivy kann in viele herkömmliche Entwicklungsumgebungen eingebettet werden, jedoch wird für die Einführung davon abgeraten, wie auch Chen und Marx (2005) in ihrer Studie verwiesen, um unnötigen Overhead durch das Erlernen der IDE-Handhabung zu vermeiden. Stattdessen sollte ein einfacher

14 http://kivy.org

Texteditor verwendet werden, der nach Möglichkeit eine Syntaxhervorhebung unterstützt. Heutzutage gibt es viele Editoren, wie Gedit, Emacs, UltraEdit, etc., die über dieses Feature verfügen. Einer davon ist Notepad++[15], der auch im Rahmen dieser Arbeit für Screenshots verwendet wurde. Er ist kostenlos, läuft jedoch nur auf Windows Betriebssystemen.

Für das Ausführen von Apps am Smartphone wird der Kivy Launcher[16] herangezogen. Dies ist ebenfalls eine App, die alle notwendigen Ressourcen beinhaltet, um Python Code direkt am mobilen Gerät ohne Installation ausführen zu können. Derzeit ist es nur für das Android Betriebssystem verfügbar und kann kostenlos von der Google Play Plattform heruntergeladen werden.

Möglichkeiten der Installation von fertigen Apps werden später in der Arbeit angesprochen und erläutert.

6 Curriculum mit Unterrichtsszenarien zur Programmiereinführung durch App-Entwicklung

In diesem Kapitel wird ein Curriculumsvorschlag für den Programmiereinstieg durch Apps-Entwicklung vorgestellt. Der Zielgruppe muss ein gewisses Abstraktionsvermögen zumutbar sein, daher eignet es sich für Personen ab schätzungsweise 14 Jahren. Das Curriculum soll die wichtigsten Programmierkonzepte sowie Basiskenntnisse der Softwareentwicklung vermitteln. Im finalen Bericht der Association for Computing Machinery[17] zu Computer Science Curricula 2013 (Armstrong, Asanovic, & Babiceanu, 2013) werden folgende Konzepte als grundlegend für die Programmierung genannt:

- Basissyntax und Semantik einer höheren Programmiersprache;
- Variablen und primitive Datentypen;
- Ausdrücke und Zuweisungen;

15 http://www.notepad-plus-plus.org
16 https://play.google.com/store/apps/details?id=org.kivy.pygame
17 www.acm.org

- einfacher Input/Output (einschließlich Dateien I/O);
- Bedingungen und iterative Kontrollstrukturen;
- Funktionen und Parameterübergabe;
- Rekursion.

An dieser Empfehlung orientiert sich auch das in dieser Arbeit vorgeschlagene Curriculum, mit geringfügigen Anpassungen, die durch den spezifischen Ansatz oder durch andere Empfehlungen aus einschlägiger Literatur bedingt sind. Einige Studien verdeutlichen beispielsweise, welche Konzepte von den Studierenden als sehr schwierig wahrgenommen werden bzw. welche Konzepte aufgrund von Best Practices mehr oder weniger für ein Einsteigercurriculum relevant sind. So werden meistens Kontrollfluss, einfache Datenstrukturen, Parameter, Geltungsbereiche, Objekte und Klassen für den Einstieg als relevant eingestuft (Schulte & Bennedsen, 2006). Auf der anderen Seite haben sich unter anderem Rekursionen, Zeiger (Lahtinen, Ala-Mutka, & Järvinen, 2005; Milne & Rowe, 2002) sowie Fehlerbehandlung und Verwendung der API als eher schwierig für Novizen herausgestellt (Lahtinen et al., 2005) und sollten daher in einem Einsteigercurriculum nicht oder nur ansatzweise herangezogen werden. Durch den 'Apps first' Ansatz wird die Curriculumsempfehlung der ACM um grundlegende objektorientierte Konzepte erweitert, da Apps von Beginn an als Klassen implementiert werden müssen. Dabei soll der Fokus jedoch auf den allgemeinen Programmierkonzepten bleiben und sich nicht auf objektorientiertes Programmieren verlagern. Des Weiteren wird die Handhabung von (Datei)Input/Output durch die App-GUI ersetzt.

Hinsichtlich der Reihenfolge, in der die erwähnten Konzepte vermittelt werden sollen, hat eine Literaturrecherche keine bestimmte Empfehlung ergeben. Angesichts des verfolgten App-Ansatzes ist die Reihenfolge jedoch ohnehin schon weitestgehend gegeben, da mit objektorientierten Konzepten und GUIs relativ früh begonnen werden muss. Für die restlichen Konzepte scheint eine Anordnung nach ihrem von Lahtinen et al., 2005 ermittelten Schwierigkeitsgrad bei den Studenten geeignet zu sein.

Somit wird der Curriculumsumfang für den Einstieg ins Programmieren durch App-Entwicklung auf folgenden Umfang und Reihenfolge festgelegt:

1. Syntax/Semantik;
2. Klassen, Methoden;
3. Variablen, Ausdrücke, Zuweisungen;
4. grundlegende Datentypen;
5. GUI/Interaktion;
6. Bedingungen;
7. Funktionen, Parameter;
8. Iterationen;
9. API und built-in Funktionen (Basiskenntnisse).

Es ist anzumerken, dass sich die Programmierkonzepte oft überschneiden bzw. nur zusammenhängend vermittelt werden können. Das kommt in einem objects first Paradigma besonders zum Vorschein, da viele Konzepte, wie Klasse, Methode, Variable, Zuweisung, etc. bereits mit dem ersten Programm enthalten sein müssen. Dem wird in der Regel versucht entgegenzuwirken, indem anfangs keine weiteren Konzepte vorgestellt werden, sondern erst die erläuterten an mehreren praktischen Beispielen verdeutlicht und befestigt werden.

Generell scheint das größte Problem bei Novizen nicht das Verständnis von den grundlegenden Konzepten zu sein, sondern deren Anwendung. Viele praktische Aufgabenstellungen mit schrittweiser Konzepterweiterung, die das 'learning by doing' fördern sollen, werden daher empfohlen (Lahtinen et al., 2005). Es wird davon ausgegangen, dass wenn die Studenten bereits einigermaßen mit dem Programmieren im Allgemeinen vertraut geworden sind und einen gewissen Grad an Selbstsicherheit erreicht haben, sie einfacher neue, komplexere Programmierkonzepte lernen können und dafür auch motivierter sein werden, sie manchmal auch auf eigene Faust zu erforschen. Eine ständige Erweiterung der Konzepte auf Basis von mangelhaftem vorherigen Wissen kann nur zu Desinteresse und Motivationsverlust führen. Des Weiteren wird beim Erlernen der Konzepte iterativ vorgegangen, d.h. die Vollständigkeit beim erläutern eines Konzepts muss nicht unbedingt gleich gegeben sein. Es werden z.B. nicht alle Datentypen gleich aufgelistet oder alle built-in Funktionen in einer Einheit bearbeitet. Es wird versucht, die Reihenfolge von Konzepten und deren Teile nach Erfordernis für die jeweilige Aufgabenstellung vorzustellen, um der Überforderung durch graduelle Vorgehensweise vorzubeugen und das Interesse aufrecht zu erhalten.

Hinsichtlich des organisatorischen Teils des Curriculums dürfte für die Einführung in das Programmieren ein Umfang von ca. 14 Wochen je 2 Stunden, mit entsprechenden Hausübungen, ausreichend sein. Es wird vorausgesetzt, dass sich die Lehrkraft ausreichend mit Python und der Kivy Library auseinandergesetzt hat, um die Basiskonzepte des Programmierens vermitteln zu können. Das Curriculum ist so konzipiert, dass es Anhaltspunkte an die Lehrperson bietet, grundlegende Konzepte anhand von Unterrichtsszenarien zu vermitteln. Es sollte jeweils einen Theorie-/Vorzeigeteil und darauf aufbauende Aufgabenstellungen beinhalten, die zum Entwurf von einfachen praktischen Apps befähigen. Beim Theorieteil soll möglichst nicht mit bereits vorgefertigtem Code gearbeitet werden sondern es soll direkt vor und mit den Studierenden programmiert werden, um bessere Lerneffekte zu erzielen (Rubin, 2013).

Da Programmierunterricht in unterschiedlichen Institutionen, wie z.B. Gymnasien, HTL-s, Universitäten, etc. stattfindet, wird versucht ein Curriculum anzubieten, das allgemein als Richtlinie angewendet werden kann, um dessen Verwendungsmöglichkeiten möglichst breit anzusetzen. Es sollte die Basis darstellen, die Abhängig von der Institution, die das Curriculum anwendet, vom vorgesehenen Stundenausmaß und Fortschritt der jeweiligen Unterrichtsgruppe, entsprechend erweitert oder angepasst werden kann. Diesbezüglich könnten etwa begleitend zum Programmiereinstieg auch die generellen analytischen und problemlösenden Denkweisen bei Studierenden gefördert werden, da dies, obwohl essenziell, als defizitär bei vielen Novizen festgestellt wurde (Holvikivi, 2010; Lahtinen et al., 2005). Logik und algorithmisches Denken werden zwar, genau wie Grundlagen der Softwareentwicklung, unausweichlich im bestimmten Maße durch das Programmieren vermittelt, können aber auch unabhängig von einer Programmiersprache bei Studenten durch geeignete Problemstellungen und Visualisierungen gefördert werden (Futschek, 2006). Dies wäre etwa parallel zu dem Programmierunterricht oder ihm vorausgehend zu empfehlen.

Im Folgenden wird eine mögliche Umsetzung des Einsteigercurriculums durch entsprechende Anhaltspunkte für das Lehrpersonal sowie Unterrichtsszenarien präsentiert und näher erläutert.

6.1 Organisatorische und theoretische Einleitung

Der Aufbau und Struktur des Unterrichts sollte erläutert werden:

- Theoretischer/praktischer Teil;
- Einzel-/Gruppenaufgaben;
- Kurze Wiederholungen zu Beginn der Unterrichtseinheiten.

Basiserläuterungen zum Programmieren (Stichworte für die Lehrperson):

- Programmieren bezeichnet die Erteilung von Anweisungen an den Rechner in textueller Form;
- Beispiele aus Realität für ein Programm (z.B. Kochrezept), Algorithmus;
- Die Funktionsweise des Rechners basiert auf dem Binärsystem, den Ziffern 0 und 1;
- Kurzer historischer Überblick über die Bedienung von Rechnern;
- Höhere Programmiersprachen wurden entwickelt, um die Kommunikation mit dem Rechner für den Menschen zu vereinfachen (Compiler, Interpreter);
- Eine Vielzahl an höheren Programmiersprachen, die anforderungsabhängig mehr oder weniger geeignet sind und entsprechend zur Anwendung kommen;
- Die Programmiersprache, die für dieses Curriculum verwendet wird, ist Python. Die Konzepte, die vermittelt werden, sind im Prinzip sprachübergreifend;
- Theoretische Einleitung zur Objektorientiertheit: die wahrnehmbare Welt kann als Sammlung von Objekten betrachtet werden, die bestimmte Eigenschaften haben und imstande sind, bestimmte Funktionalitäten oder Handlungen durchzuführen. Programme können als eine Art virtueller Realität gesehen werden – also ebenfalls als eine Ansammlung von Objekten. Skizzen zur Veranschaulichung wären geeignet.

6.2 Installation von Tools, erster Programmaufruf, Code lesen

Im nächsten Schritt erfolgt die Installation von erforderlichen Werkzeugen – Python und Kivy-Library – samt entsprechenden Erläuterungen (Python Basisinstallation mit Interpreter, Kivy – open source Python library für Entwicklung von Applikationen). Detaillierte Anleitungen können

Betriebssystemspezifisch online nachgeschlagen werden[18]. Des weiteren wird ein geeigneter Texteditor (z.B. Notepad++) benötigt, in dem das eigentliche Programmieren stattfinden wird.

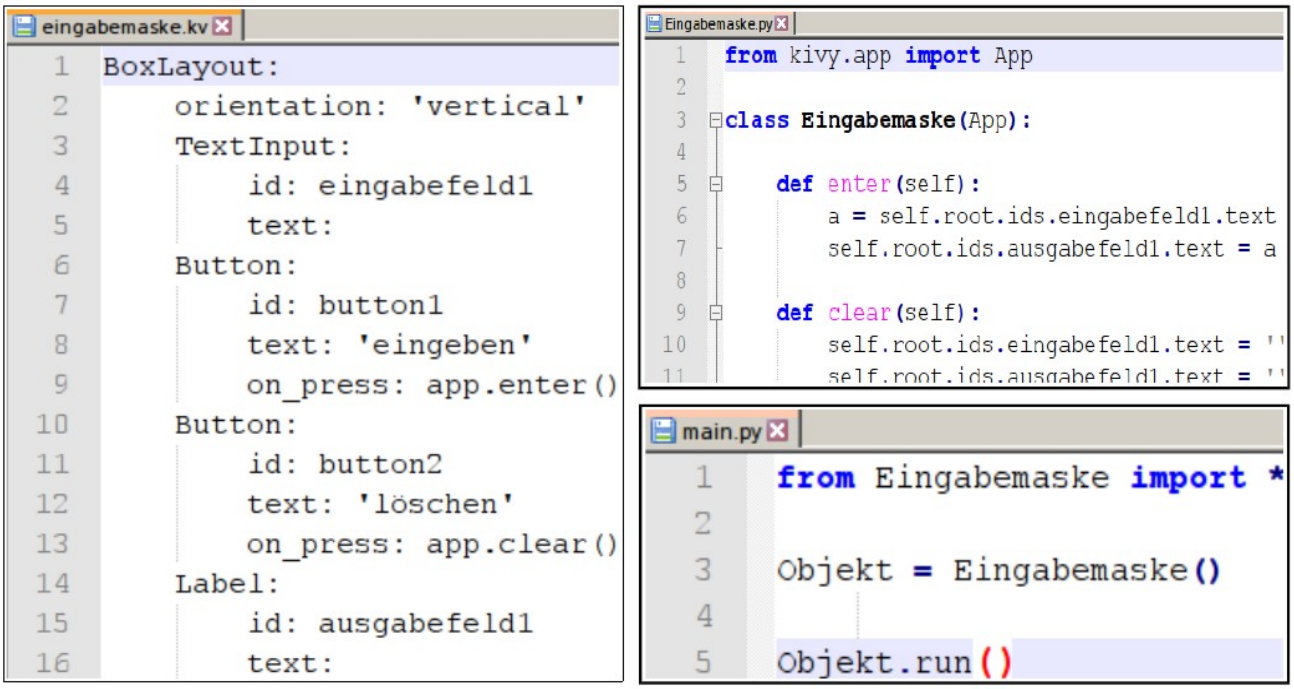
Obwohl das Verfassen von Python-Quellcode in jedem beliebigen Texteditor möglich ist, wird die Verwendung von Editoren empfohlen, die eine Python-Syntax Hervorhebung eingebaut haben. Eventuell kann zu Beginn ein einfacher Texteditor verwendet werden, um zu verdeutlichen, dass Code nur eine Sammlung von Textzeilen ist, was durch die farbliche Kennung womöglich missdeutet werden könnte.

6.2.1 Der erste Kontakt mit Code

In der Studie „Objects from the beginning – with GUIs“ (Proulx, Raab, & Rasala, 2002) wird argumentiert, dass Studenten einfache Algorithmen verstehen und das generelle Verständnis von Objekten aus der Realität bereits haben. Darauf aufbauend sollten sie gleich mit Code beginnen, die einfache Klassen mit Anwendungsbezug darstellen. Auch wenn sie die Details der Sprachelemente oder die genaue Syntax nicht verstehen - ein generelles Gefühl und Überblick über die Konzepte kann schon vermittelt und graduell verfeinert werden. Ähnlich wie beim Erlernen einer menschlichen Sprache, sollten Studenten daher möglichst viel mit Code konfrontiert werden, wobei Programmierkonzepte parallel erlernt werden.

Folgend der Studie könnte ein erstes Beispiel im Rahmen eines Curriculums zur Einführung in die Programmierung durch App-Entwicklung etwa wie in Abbildung 5 ausschauen.

18 http://kivy.org/#download



eingabemaske.kv

```
BoxLayout:
    orientation: 'vertical'
    TextInput:
        id: eingabefeld1
        text:
    Button:
        id: button1
        text: 'eingeben'
        on_press: app.enter()
    Button:
        id: button2
        text: 'löschen'
        on_press: app.clear()
    Label:
        id: ausgabefeld1
        text:
```

Eingabemaske.py

```
from kivy.app import App

class Eingabemaske(App):

    def enter(self):
        a = self.root.ids.eingabefeld1.text
        self.root.ids.ausgabefeld1.text = a

    def clear(self):
        self.root.ids.eingabefeld1.text = ''
        self.root.ids.ausgabefeld1.text = ''
```

main.py

```
from Eingabemaske import *

Objekt = Eingabemaske()

Objekt.run()
```

Abbildung 5: Der Quellcode zur ersten App (Eingabemaske)

Die Dateien **eingabemaske.kv**, **Eingabemaske.py** und **main.py** werden den Studierenden zu Beginn vorgefertigt zur Verfügung gestellt. Diese müssen in ein lokales Verzeichnis auf ihren Rechnern kopiert werden. Im Anschluss soll der erste Programmaufruf getätigt werden. Dafür wird ein Konsolenfenster geöffnet, zum Kivy-Installationsverzeichnis gewechselt (falls der Pfad für Python durch eine Umgebungsvariable nicht bereits gesetzt wurde) und folgender Befehl abgesetzt:

python <Pfad bis zum Verzeichnis mit Beispielen>\Beispiel1\main.py

Ein Fenster wie in Abbildung 6 sollte erscheinen (beim Aufruf ist das Eingabefeld leer).

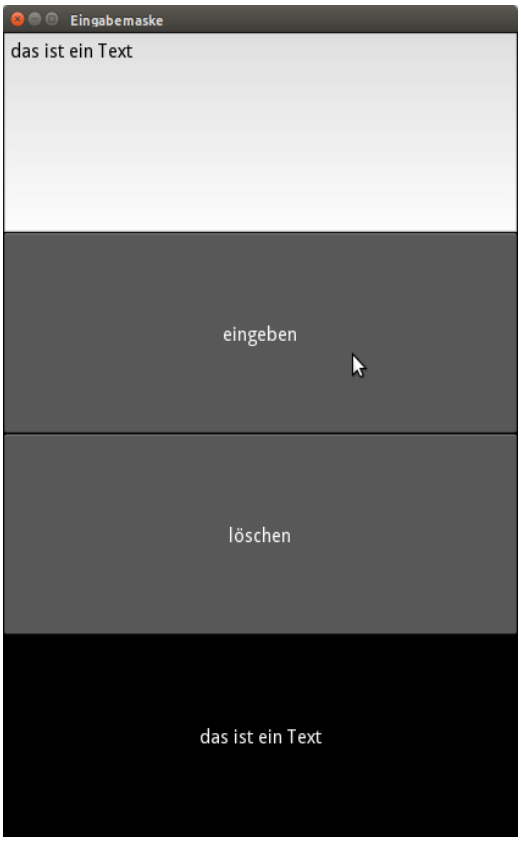


Abbildung 6: Erstes Beispiel: App-Aufruf

Nachdem die Maske sichtbar geworden ist, sollten die Quellcodedateien geöffnet und besprochen werden.

Kölling und Rosenberg (2001) geben in ihrer Studie Richtlinien für die Vermittlung von OO-Konzepten für Programmieranfänger, wobei diese im Grunde unabhägig von einer bestimmten Sprache angewendet werden können. An erster Stelle sollen lt. den Authoren Objekte erstellt und Methoden aufgerufen werden. Es soll kein Code geschrieben, sondern ein generelles Verständnis von Code und dessen Struktur/Syntax vermittelt werden. Der erste Schritt soll also das Lesen von Code sein, und nicht Codieren selbst.

Daraus ergibt sich die erste Aufgabe – nämlich die lokal am Rechner ausgeführte App mit unterschiedlichen Eingaben zu testen sowie das Verhalten der GUI-Elemente zu beobachten und Rückschlüsse auf deren Funktionsweise zu ziehen. In Folge soll der zugrundeliegende Code parallel mit der ausgeführten App besprochen und analysiert werden.

6.2.2 Syntax, Klasse, Methoden, Variablen, Zuweisung

An diesem relativ einfachen Beispiel können bereits viele unterschiedliche Programmierkonzepte, die im folgenden aufgelistet sind, illustriert werden. Die Objektorientiertheit wird dabei von Beginn an vermittelt.

- **Syntax/Semantik:** Eine Programmiersprache enthält wie eine natürliche Sprache (Schlüssel)wörter mit bestimmter Bedeutung, die nach festgelegten Regeln angeordnet werden müssen.
- **Gliederung**: Es ist ersichtlich, dass der Code sowohl in der .py als auch der .kv Datei eine gewisse Gliederung hat (Python Einrückungen), und dass die jeweiligen Zeilen zusammenhängend betrachtet werden sollen (ähnlich der Verzeichnisstruktur eines Betriebssystems).
- **Trennung von GUI und Programmlogik:** Es ist ersichtlich, dass die GUI und die Programmlogik in unterschiedlichen Dateien strukturiert sind. Die kv-Sprache für die GUI ist sehr intuitiv und braucht in diesem Beispiel relativ wenig Erläuterung.
- **Interaktion**: Die Kommunikation zwischen der GUI, Programmlogik und dem Programmaufruf ist bereits im ersten Beispiel gegeben und sollte ausführlich erläutert werden.
- **Objektorientiertheit**: Die Eingabemaske wird als eine Code-Einheit (Klasse) wahrgenommen, was die objektorientierte Denkweise von vornhinein fördern soll. Die Instanziierung und der Programmaufruf erfolgt in einer separaten main-Datei, damit die Klassendatei 'rein' vom nicht klassenrelevantem Code bleibt, um Verwirrungen zu vermeiden.
- **Methoden:** Die objektorientierten Konzepte und Begriffe sollen schon von Beginn an eingeführt und erläutert werden.
- **Sequenzielle Abarbeitung von Codezeilen**: im Rahmen der Methoden werden die Anweisungen in den Zeilen nacheinander ausgeführt und haben eine bestimmte Reihenfolge.
- **Variable**: Bereits im ersten Beispiel wird eines der Basiskonzepte der Programmierung – die Variable – vorgestellt. Eine Anknüpfung an das vorhandene Wissen aus der Mathematik sollte in Erwägung gezogen werden.
- **Zuweisung**: Der Zuweisungsoperator wird ebenfalls im ersten Beispiel vorgestellt. Im späterenVerlauf des Curriculums wird er vom Vergleichsoperator abgegrenzt.

Die erwähnten Programmierkonzepte sollten anhand der verwendeten Qellcodedateienl erläutert werden. Einige Stichworte für den/die LehrerIn bez. des Aufbaus der Dateien:

1. *main.py*
    - Nur für die Initialisierung des Programms zuständig und kann in weiterer Folge außer Acht gelassen werden;
    - Die Aufrufdatei ist eine Voraussetzung für das Starten von Apps am Smartphone.
2. *eingabemaske.kv*
    - Für die grafische Darstellung zuständig;
    - Der Dateiname ist dem Klassennamen gleich, nur kleingeschrieben;
    - Es wird ein bestimmtes Layout und Orientierung (horizontal/vertikal) gesetzt;
    - Beinhaltet grafische Elemente – sog. Widgets (Button, Label, TextInput, etc.). Diese haben bestimmte Eigenschaften (ID, Beschriftung, Funktion usw.).
3. *Eingabemaske.py*
    - Für die Verarbeitungslogik zuständig;
    - Klasse – abstraktes Modell eines Objektes, das mit bestimmte Funktionen (Methoden) versehen ist;
    - Erwähnung, ohne ins Detail zu gehen, der fix vorgegebenen Codeteile, die immer vorhanden sein müssen und mit copy-paste und entsprechenden Anpassungen für alle Programme verwendet werden können (import statement, App-Klassen Ableitung und deren Aufruf).

In der vorigen Einheit wurde die Objektorientiertheit theoretisch erläutert. An dieser Stelle soll das Konzept nochmal aufgegriffen und anhand Klassen erklärt werden: Mit einer Klasse präzisieren wir dem Rechner, was wir unter einem bestimmten Konstrukt, das wir in unserem Programm verwenden wollen, verstehen. Wir haben also mit dem Code definiert, was eine Eingabemaske ist bzw. tut. Noch haben wir keine Eingabemaske „erzeugt“, das geschieht erst in der main.py - Datei. Eine Klasse kann man sich als eine Art Schablone vorstellen, mithilfe welcher entsprechende Objekte erstellt werden können.

Für ein umfassendes Verständnis vom objektorientierten Programmieren eignen sich wohl am besten Programme, die mehrere unterschiedliche Klassen und Subklassen beinhalten, welche auch untereinander interagieren, um bestimmte Aufgaben zu erzielen (Kölling & Rosenberg, 2001). Idealerweise sollten die Klassen auch eine graphische Repräsentation aufweisen, um sie begreifbarer für Novizen zu gestalten (Proulx et al., 2002). Im Rahmen von Apps ist dies zwar auch möglich, jedoch für ein Einstiegscurriculum eher nicht geeignet, da, abseits von gängigen Widgets, graphische Darstellungen auf der GUI die Instanziierung und Verwendung von komplexeren Klassen benötigen, was für Verwirrung bei Programmieranfängern sorgen dürfte. Daher sollten für den Einstieg die Vermittlung von grundlegenden Begriffen der OOP-Konzepte reichen, was ohnehin durch Erzeugung von bereits einfachsten Apps gegeben ist, da nur als Klasseninstantiierung realisierbar. Des Weiteren ist die GUI-Implementierung nichsts anderes als Umgang mit Klassenobjekten (Widgets) und deren Klassenattributen (Eigenschaften) sowie Mehoden, was implizit die Objektorientiertheit vermittelt. Ein tiefes Verständnis der Zusammenhänge benötigt jedoch seine Zeit, und soll durch Anwendung an Beispielen und Aufgeben befestigt werden. In diesem Stadium sollte nur das Grundlegende zum Verständnis des ausgeführten Codes (Abbildung 5) angesprochen werden. Eine tiefere Erläuterung von weiteren objektorientierten Konzepten wäre nicht zielführend, da Studenten bereits mit dem Verstehen der bloßen Syntax gefordert sind.

Die *self* bzw. *self.root.ids* Syntax ist zu Beginn wohl am schwierigsten zu verstehen, was aber kein Hindernis darstellen sollte, da manche Elemente (vorerst) einfach als gegeben betrachtet werden müssen. Darunter fallen z.B. das Importstatement, das Subclassing (App), sowie die self-Parameterübergabe in jeder Methode.

Die drei Schlüsselwörter *self.root.ids* werden immer wieder zusammenhängend im Code verwendet, um Elemente in der GUI zu referenzieren. Danach folgt die ID des Elements, welches referenziert werden soll, sowie die Eigenschaft, die mit der Anweisung bearbeitet wird.

Self bedeutet, dass es sich bei der jeweiligen Anweisung um die Bearbeitung von Elementen des aufrufenden Klassenobjekts handelt (äquivalent zu this in Java).

Root bezeichnet die GUI der jeweiligen Klasse, genau genommen das erste Element der .kv Datei, unter dem alle anderen graphischen Elemente hierarchisch angeordnet sind. All diese Elemente müssen eindeutig referenzierbar sein, falls sie eine interaktive Funktionalität enthalten sollen, und das geschieht durch die Vergabe von IDs (Identifiern).

Somit kann beispielsweise das Konstrukt *self.root.ids.ausgabefeld1.text* wie folgt interpretiert werden (Abbildung 7):

- self. referenziere ein Element der aufrufenden Klasse (was der Regelfall sein wird)
- root. referenziere danach deren (der aufrufenden Klasse) GUI
- ids. referenziere im Rahmen der GUI die enthaltenen IDs (als Sammlung)
- ausgabefeld1. referenziere in der ID-Sammlung die Element-ID 'ausgabefeld1'
- text referenziere die Eigenschaft 'text' des ausgewählten Elements

Des Weiteren ist wichtig zu betonen, dass die Referenzierung der Programmierlogik aus der GUI über das Schlüsselwort *app.* zu erreichen ist.

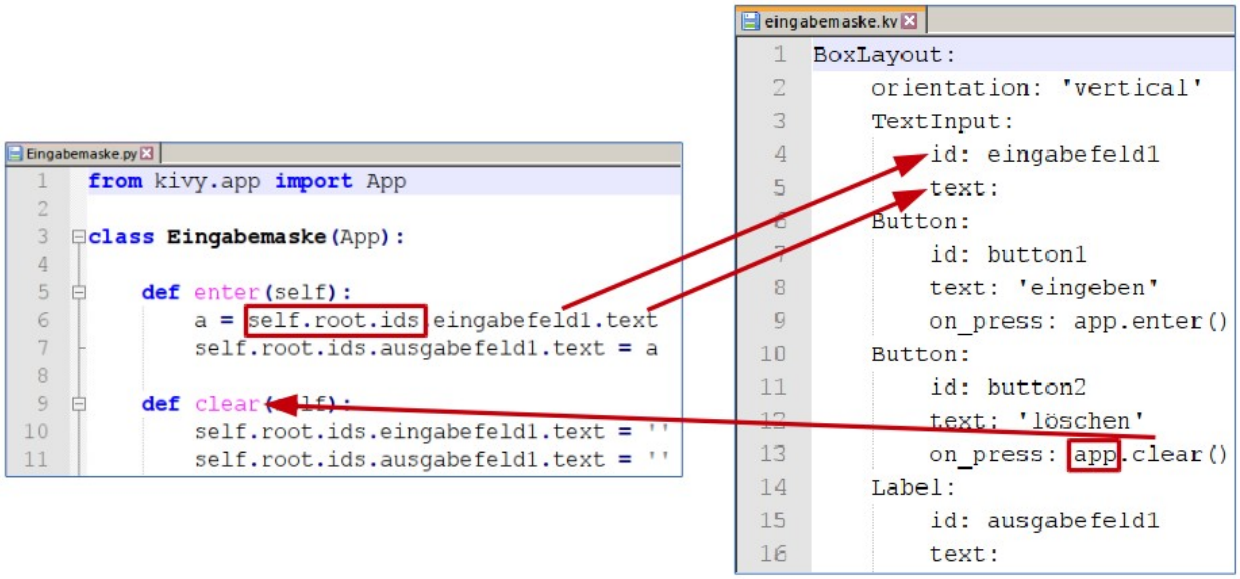


Abbildung 7: Veranschaulichung der Codeinteraktion zwischen App und GUI

6.2.3 Übertragung der App zum Smartphone

Der nächste Schritt sollte sein, die am Rechner getestete App zum Smartphone zu übertragen und ernuete Tests, diesmal am Handy durchzuführen. Gleich am Anfang soll diese motivationsfördernde Wirkung des selbstgeschaffenen „greifbaren“ Fertigprodukts geschöpft werden. Im Rahmen dieser

Diplomarbeit wurde zum Testen der Apps ein auf Android basiertes Smartphone verewendet. Der einfachste Weg, Apps auf Mobilgeräten mit diesem System auszuführen ist in Anhang 8.1.1 beschrieben. Für andere Betriebssysteme siehe Anhang 8.1.2.

6.3 Geringfügige Codeanpassungen

Wiederholung: In den ersten Einheiten des Curriculums kann die Wiederholung sowie Fragen und Antworten einen größeren Teil des Unterrichts einnehmen, da Studierende mit sehr vielen Konzepten bereits am Anfang konfrontiert wurden und deren grundlegende Funktionsweise essenziell für den weiteren Verlauf des Curriculums ist. Es sollte ausführlich alles aus der vorigen Stunde wiederholt werden und möglichst alle relevanten Fragen geklärt werden bevor zum nächsten Schritt gewechselt wird.

Als zweite Richtlinie schlagen Kölling und Rosenberg (2001) vor, kleinere Änderungen am bestehenden Code und Tests vorzunehmen, um sich mit den Programmierregeln einigermaßen vertraut zu machen. Der Zyklus von Änderung/Ausführung/Errorhandling wird ebenfalls dadurch vermittelt. Die Änderungen sollen graduell von trivialen zu komplexeren erfolgen.

6.3.1 Variablen, Strings und Stringmanipulationen

Im theoretischen und Vorzeigeteil dieser Einheit können Variablen und Strings als ein einfacher Einstieg in die Codierung im Sinne von Änderungen vom vorhandenen Quellcode herangezogen werden. Folgende Schlagwörter können als Richtlinien für die Lehrperson dienen:

- Erläuterung des Variablenbegriffs. Zuweisung eines Werts / einer Variable. Sinnvolle Benennung von Variablen(Regeln für Python beachten![19]). Variablen verknüpfen.
- Erläuterung vom Datentyp String. Alle Ein-/Ausgaben in der App werden als String interpretiert. Manipulationen mit Strings veranschaulichen.

Beispiele, die für die erwähnten Konzepte herangezogen werden könnten (Methodenausschnitte als Abhilfe für den/die LehrerIn):

19 Variablen müssen z.B.mit _ oder einem Buchstaben beginnen, sind case-sensitive, etc.

```
def enter(self):
      texteingabe = self.root.ids.eingabefeld1.text
      textausgabe = texteingabe
      textausgabe = 'Neue Wertzuweisung überschreibt die alte'
      self.root.ids.ausgabefeld1.text = textausgabe
```

Verdutlichung von:

- Beliebigkeit der Variablennamen;
- deren möglichst sinnvolle Benennung;
- Änderungen/Überschreibungen von Variablen;
- Zuweisung des Variablenwertes von rechts nach links.

Textverknüpfungen:

```
textausgabe = 'Ihre Eingabe lautet: ' + texteingabe
textausgabe = texteingabe + ' , lautet Ihre Eingabe'

textausgabe = texteingabe + texteingabe ' , lautet Ihre Eingabe 2 mal'
textausgabe = texteingabe * 2 + ' , lautet Ihre Eingabe 2 mal'

texteingabe = texteingabe + texteingabe #Zuweisung der Variable sich selbst!
textausgabe = texteingabe + ' , lautet Ihre Eingabe 2 mal'
```

Verknüpfung langer Zeilen durch Klammern oder Backslash:

```
textausgabe = texteingabe + (' … Übrigens: Texte können auf mehrere Zeilen '
                             'verteilt werden '
                             'wenn sie in Klammern gesetzt werden...')

textausgabe = texteingabe + ' Mehrzeiliger Code kann auch durch ' + \
                             'Backslash erzielt werden..'
```

Textumbruch erzwingen durch \n:

```
textausgabe = texteingabe + (' Zeilenumbruch wird durch Backslash-n \n'
                             'in der Anzeige erzielt..')
```

Textindizierung (indexing):

```
texteingabe = # z.B. 'Hallo Welt!'
textausgabe = texteingabe[0] #H
textausgabe = texteingabe[4] #0
textausgabe = texteingabe[-1] #! (letzter Wert)
```

Textausschnitte (slicing):

```
texteingabe # z.B. 'Hallo Welt!'
textausgabe = texteingabe[0:5] #Hallo
textausgabe = texteingabe[5:9] # Wel

texteingabe = '123456789'
textausgabe = texteingabe[0:9:2] # 13579
```

Neben den vorgeschlagenen können natürlich auch andere Stringmanipulationen herangezogen werden, die im Anschluss durch Einzelaufgaben geübt werden sollen. Studierende sollen sich durch kleine Anpassungen möglichst viel mit Code an sich auseinandersetzen und sich mit der generellen Struktur und Programmierlogik vertraut machen. Des Weiteren soll ihnen gleich das Gefühl vermittelt werden, dass sie selbst bereits programmieren, bzw.einen Einfluss auf die Programmfunktionalität haben, was durch die Visualisierung in der App-GUI in einem noch stärkeren Ausmaß gegeben sein sollte.

Für den praktischen Teil des Unterrichts sollte der vorhandenen Code der Eingabemaske (Abbildung 5) entsprechend den Vorgaben der Lehrperson angepasst und getestet werden. Einige einfache Beispiele könnten sein:

- als Eingabe wird der Name des jeweiligen Benutzers erwartet, die Ausgabe soll lauten 'Hallo'+ eingegebener Name, z.B.:

```
def enter(self):
      a = self.root.ids.eingabefeld1.text
      self.root.ids.ausgabefeld1.text = 'Hallo' + a
```

- die Aufgabe kann variiert werden. Lösungen dazu könnten wie folgt ausschauen:

```
def enter(self):
      a = 'Hallo' + self.root.ids.eingabefeld1.text
      self.root.ids.ausgabefeld1.text = a

def enter(self):
      a = 'Hallo'
      b = self.root.ids.eingabefeld1.text
      self.root.ids.ausgabefeld1.text = a + b
```

```
def enter(self):
      vorname = self.root.ids.eingabefeld1.text
      grat = ', Gratuliere zu deinem ersten Programm!'
      self.root.ids.ausgabefeld1.text = 'Hallo ' + vorname + grat
```

Eine zu Beginn etwas fortgeschrittene Aufgabe könnte beispielsweise sein, das Alphabet in eine Variable als String zu speichern und anhand indexing/slicing den eigenen Vor- und Nachnamen auszugeben.

Des Weiteren sollte auch hinsichtlich der GUI des Codes darauf hingewiesen werden, was fest vorgegebene Schlüsselwörter sind, welche Elemente geändert werden können und welche Änderungen der Klasse dadurch impliziert werden, z.B. (vergleiche auch Code in Abbildung 5):

```
Button:
    id: button_clear
    text: 'löschen'
    on_press: app.bereinigen()
```

```
def  bereinigen(self):
  self.root.ids.eingabefeld1.text = ''
  self.root.ids.ausgabefeld1.text = ''
```

Während des Unterrichts sollte auch auf die übersichtliche Schreibweise von Code hingewiesen werden[20], z.B.:

- Leerzeichen vor und nach Gleichheitszeichen (nicht jedoch in Klammern – bei Methoden z.B.);
- Einrückungen um jeweils 4 Leerzeichen;
- Codezeilen auf max. 80 Zeichen beschränken, Klammern und '\' für Fortsetzung des Codes in nächster Zeile verwenden;
- Verwendung von Kommentaren für Erklärungen (und Dokumentation).

Das zentrale Ziel dieser Unterrichtseinheit sollte sein, ein generelles Gefühl für den Umgang mit Code zu vermitteln, sowie die Unterscheidung zwischen den vorgegebenen, fixen und den veränderlichen Codeteilen hervorzuheben.

6.4 GUI Codeanpassungen

Der Fokus der Modifikationen bei den ersten Einheiten sollte jedoch auf der GUI sein, da deren Anpassungen intuitiv verständlich sind und visuell ersichtliche Änderungen nach sich ziehen, was den Eindruck einer großen Modifikation und bereits einer aktiven Programmgestaltung verleihen dürfte. Des Weiteren wird die richtige Vorgehensweise vermittelt – zuerst wird die Schnittstelle entworfen, was der Benutzer sehen soll, und dann die zugrundeliegende Funktionalität implementiert.

Im theoretischen Teil sollen die Grundlagen der GUI am praktischen Vorzeigen dargestellt werden;

- Layouts als Anordnungsmodell für enthaltene visuelle Elemente erläutern. Auf die wichtigsten, die verwendet werden, näher eingehen:
  - BoxLayout: horizontale/vertikale Anordnung von enthaltenen Elementen (Abbildung 8);

20 Ausführliche Richtlinien können unter https://www.python.org/dev/peps/pep-0008/ abgerufen werden. Hier sind nur die Wichtigsten zu erwähnen.

```
BoxLayout:
      orientation: 'horizontal'
      TextInput:
            id: eingabefeld1
            text:
      Button:
            id: button1
            text: 'eingeben'
            on_press: app.enter()
      Button:
            id: button2
            text: 'löschen'
            on_press: app.clear()
      Label:
            id: ausgabefeld1
            text:
```

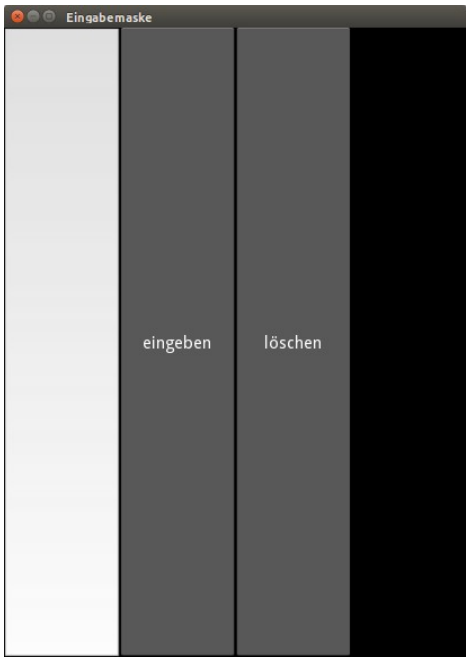


Abbildung 8: Layouts: Box - horizontal

- GridLayout (Abbildung 9): Anordnung von Elementen im Raster. Anzahl der Spalten(*cols*) oder Zeilen(*rows*) müssen angegeben werden;

```
GridLayout:
      cols: 2
      rows: 2
      TextInput:
            id: eingabefeld1
            text:
      Button:
            id: button1
            text: 'eingeben'
            on_press: app.enter()
      Button:
            id: button2
            text: 'löschen'
            on_press: app.clear()
      Label:
            id: ausgabefeld1
            text:
```

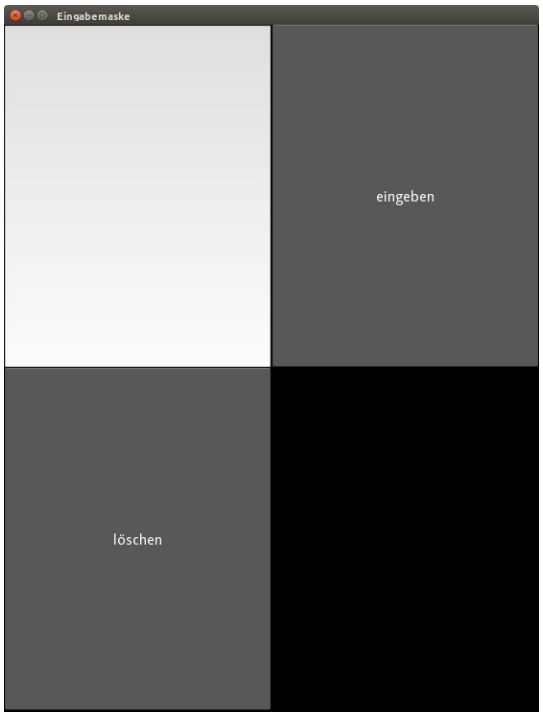


Abbildung 9: Layouts - Grid

- Verschachtelung von Layouts (Abbildung 10) ist auch möglich (Einrückungen dabei beachten!);

```
BoxLayout:
    orientation: 'vertical'
    TextInput:
        id: eingabefeld1
        size_hint: 0.75,0.4
        text:
    TextInput:
        id: eingabefeld2
        size_hint: 1,0.5
    GridLayout:
        cols: 3
        Button:
            id: button1
            text: 'eingeben'
            background_color: 0.2,0.2,0.8,1
            on_press: app.enter()
        Button:
            id: button2
            text: 'löschen'
            on_press: app.clear()
        Label:
            id: ausgabefeld1
            text:
```

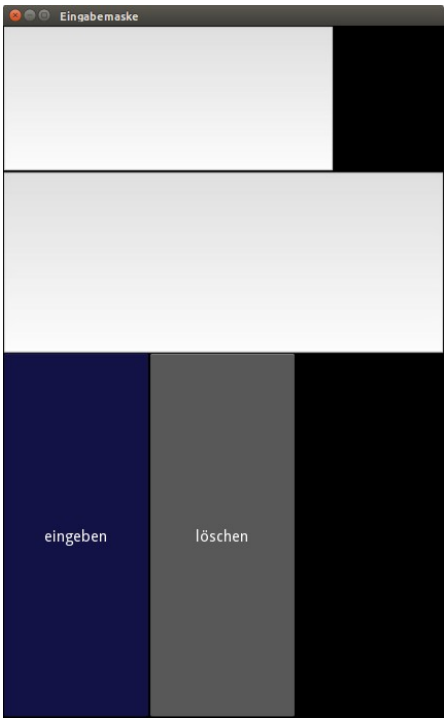


Abbildung 10: Layouts - verschachtelt

- andere Eigenschaften der GUI-Elemente heranziehen, wie z.B. die Größe und Position des Textes, dessen Farbe oder Hintergrundfarbe ändern. Readonly, multiline (meistverwendete Attribute behandeln). Ids können bei Widgets, die nicht referenziert werden müssen, ausgelassen werden oder können nachträglich definiert werden;
- padding (Abstand vom Displayrahmen);
- spacing (Abstand von Elementen zwischeneinander);
- size_hint: x,y – proportionale Aufteilung der verfügbaren Fläche (Dezimalzahl 0-1);
- Labels:
  - text_size;
  - halign: horizontale Ausrichtung. Werte: left, center, right, justify. Standard: 'left';
  - valign: vertikale Ausrichtung: Werte: bottom, middle, top. Standard: 'bottom';

- bold, italic: (True/False);
- color: Textfarbe im Format R,G,B,A (red, green, blue, alpha – Transparenz) Werte zwischen 0 und 1, Standardmäßig auf 1,1,1,1 (weiß) gesetzt. An dieser Stelle ist womöglich ein kleiner Exkurs angebracht, um farbliche Darstellungen im Digitalformat am Rechner zu erklären.[21]

- Buttons:
  - erbt Eigenschaften von Label;
  - text: Beschriftung;
  - background_color: Farbe des Buttons im RGBA Format (siehe Label);
  - disabled: erbt von Widget (Oberklasse).
- TextInput:
  - background_color: Farbe des Buttons im RGBA Format (siehe Label);
  - disabled: erbt von Widget (Oberklasse);
  - font_size: Textgröße in Pixel. Standard 10;
  - hint_text: Hinweistext wenn Feld leer und kein Fokus darauf;
  - hint_text_color: Farbe (RGBA) vom Hinweistext;
  - input_filter: erlaubte Eingabe. Werte: None, int, float. Standard: None;
  - multiline: True/False. Standard: True;
  - password: True/False. Standard: False;
  - readonly: True/False. Standard: False.

Die erwähnten Eigenschaften der jeweiligen meistverwendeten Widgets und ev. andere, die in der API[22] zu finden sind, sollen erläutert und durch Codierung vor den Studenten veranschaulicht werden. Auf

21 Ein online Farbengenerator kann verwendet werden, z.B. http://www.w3schools.com/tags/ref_colorpicker.asp

22 http://kivy.org/docs/api-kivy.html

die variabilität der Namen und die sinngemäße Bezeichnungswahl sowie ID-Vergabe sollte im Rahmen des Vorzeigeteils hingewiesen werden.

Darauf aufbauend sollen entsprechende Aufgaben vergeben werden, um die GUI-Kenntnisse zu befestigen. Es können beispielsweise Skizzen von Anforderungen über Layout, Widgets (Buttons, Labels, TextInput) mit genau vorgegebenen Eigenschaften (Position, Größe, Beschriftung, Farbe, etc.) angefertigt werden, die von Studierenden als Laborübung und / oder Hausübung implementiert werden sollen. In dieser Phase werden noch keine neuen Methodenaufrufe den Buttons zugewiesen. Es soll erläutert werden, dass deren Implementierungen im Nachhinein stattfinden werden und dass diese Einheit nur die GUI berücksichtigt.

6.5 Funktionelle Code-Änderungen

Die folgenden Schritte in einem Einsteigercurriculum sollten laut Kölling und Rosenberg (2001) funktionelle Änderungen von bestehenden bzw. die Erstellung eigener einfachen Methoden sein. Aufbauend auf den GUI-Übungen und Anpassungen aus der vorigen Lehreinheit, sollten daher im nächsten Schritt einige neue Funktionalitäten von Buttons schrittweise implementiert werden. Als Beispiel könnte die Maske in Abbildung 11 herangezogen werden. Diese könnte etwa im Rahmen der Hausübung zur letzten Einheit als Erweiterung der ursprünglichen Eingabemaske zum programmieren aufgegeben worden sein. Nun sollten deren Methodenaufrufe per Buttonklick einzeln vor den Studenten im Vorzeigeteil dieser Einheit implementiert werden, bis etwa der nachfolgende Code entsteht:

```
from kivy.app import App

class Eingabemaske(App):

  def enter1(self):
    input = self.root.ids.eingabefeld1.text
    self.root.ids.ausgabefeld1.text = input

  def enter2(self):
    input = self.root.ids.eingabefeld2.text
    self.root.ids.ausgabefeld1.text = input

  def merge(self):
    prefix = 'Die Verknüpfung lautet: '
    input1 = self.root.ids.eingabefeld1.text
    input2 = self.root.ids.eingabefeld2.text
    self.root.ids.ausgabefeld1.text = prefix + \
              input1 + input2

  def add(self):
    prefix = 'Die Summe lautet: '
    input1 = int(self.root.ids.eingabefeld1.text)
    input2 = int(self.root.ids.eingabefeld2.text)
    summe = input1 + input2
    self.root.ids.ausgabefeld1.text = prefix + \
              str(summe)

  def clear_eingaben(self):
    self.root.ids.eingabefeld1.text = ''
    self.root.ids.eingabefeld2.text = ''

  def clear_ausgabe(self):
    self.root.ids.ausgabefeld1.text = ''
```

Abbildung 11: Eingabemaske - erste Funktionalitätsanpassungen

Die graduelle Vorgehensweise soll auch explizit den nicht-linearen sondern iterativen Prozess der Programmierung verdeutlichen, bei dem der Code immer wieder getestet und schrittweise erweitert/vervollständigt wird.

6.5.1 Weitere Datentypen, Casting und arithmetische Operatoren

Es wird empfohlen, vorerst einfachere Themengebiete wie die Arithmetik im Curriculum durchzunehmen, die leicht verständlich bzw. bereits bekannt sind, damit sich Studierende auf die Programmstruktur konzentrieren und sich damit mehr vertraut machen können. In den Methodenerweiterungen würden sich daher nun etwa die Programmierkonzepte weiterer Datentypen, Casting und arithmetischen Operatoren eignen. Diese sollten anhand von Beispielen, wie im Code zur Maske in Abbildung 11 vorgezeigt und näher erläutert werden.

- Einführung von Datentypen Int und Float neben String;
- die built-in Arithmetik (+, -, *, /, %, **) soll verdeutlicht werden. Auf Unterschiede bei der Verwendung von int/float eingehen (Abschneiden von Dezimalzahlen, z.B. 7/3 = 2);
- deren Umwandlung mit Casting und entsprechende Regeln/Begrenzungen diesbezüglich sollen mit Beispielen erläutert werden:
  - nach string können alle Datentypen umgewandelt werden;
  - nach int/float können nur dem Datentyp entsprechende Strings umgewandelt werden;
  - beim casting von float nach int werden die Zahlen nach dem Dezimaltrennzeichen abgeschnitten.

Beispielsweise können an einer Methode, welche den Umfang eines Rechtecks berechnet, mehrere erwähnte Konzepte erläutert werden. Mit entsprechenden Anpassungen sind die Unterschiede und das Casting zwischen Int und Float leicht zu verdeutlichen:

```
def rechteck_umfang(self):
      laenge = float(elf.root.ids.eingabefeld1.text)
      breite = float(self.root.ids.eingabefeld2.text)
      umfang = (laenge + breite) * 2

      ausgabe = 'Der Umfang des Rechtecks mit Länge ' + str(laenge) +\
              ' und Breite ' + str(breite) + ' beträgt: ' + str(umfang)

      self.root.ids.ausgabefeld1.text = ausgabe
```

Im Anschluß kann durch Aufgaben das bislang Erlernte befestigt und mit dem neuen Wissen verknüpft werden. Beispielsweise könnte ein GridLayout mit mehreren Buttons unterschiedlicher Farben und Textgrößen versehen werden, die jeweils eine andere arithmetische Operation ausführen.

6.6 Debugging, Bedingungen

6.6.1 Ausnahme- und Fehlerbehandlung, Kommentare

Während des Testens der Eingabemaske in Abbildung 11 mit dem vorgeschlagenen Code dürften wahrscheinlich Abstürze bemerkt worden sein, falls z.B. keine (ganze) Zahlen beim Betätigen des Buttons 'Zahlen addieren' eingegeben wurden. In der Konsole erscheinen in dem Fall bestimmte Fehlermeldungen, auf die eingegangen werden sollte (siehe z.B. Abbildung 12). Es wird empfohlen, die Fehlerbehandlung relativ bald im Curriculum vorzustellen, da es sich um die Entwicklung von Smartphone-Apps handelt, die unmittelbar auf den mobilen Geräten der Studierenden übertragen und benutzt werden sollen. Deren Abstürze sollten möglichst verhindert werden, um Frustrationen und negativen Auswirkungen auf die Motivation für den Programmierunterricht vorzubeugen.

```
    on_press: app.add()
  File "main.py", line 22, in add
    b = int(self.root.ids.eingabefeld1.text)
ValueError: invalid literal for int() with base 10: 'x'
```

Abbildung 12: Beispiel einer Fehlermeldung verursacht durch unverträgliche Datentypen

In der Regel wird die Methode samt Codezeile und Fehlerhinweis ausgegeben, der entsprechend behandelt werden sollte. Unterschiede zwischen den Fehlertypen sollte erläutert werden. Z.B.:

- Syntaxfehler: `a = 'Wort` (Anführungszeichen am Ende fehlt, Programm wird gar nicht ausgeführt) – Fehlermeldung mit ^ an der problematischen Stelle;
- Semantikfehler: `c = '3' + '4'` (Ergebnis: 34 statt 7, vorausgesetzt eine Addition war erwünscht. Programm wird ausgeführt, liefert aber nicht den gewünschten output);
- Laufzeitfehler:

```
a = self.root.ids.eingabefeld1.text
#Eingegeben wird ein Text, z.B. 'Hallo'
b = int(a)
```

  Programm wird ausgeführt, stürzt aber ab, wenn nicht der erwartete Datentyp (ganze Zahl) eingegeben wird.

Die Vorgehensweise beim Auftreten des zuletzt erwähnten Fehlertyps ist zu erläutern, damit Applikationsabstürzen vorgebeugt wird. Das try-catch Konzept für das Abfangen von Fehlern sollte an einigen Beispielen vorgestellt werden.

Es könnte etwa die bekannte Regel herangezogen werden, dass eine Division durch Null nicht erlaubt ist. Die Ausführung folgender Beispielmethode würde zum Applikationsabsturz mit einer Fehlermeldung wie in Abbildung 13 führen:

```
def divideNull(self):
    self.root.ids.ausgabefeld1.text = 3/0
```

```
    self.root.ids.ausgabefeld1.text = 3/0
ZeroDivisionError: integer division or modulo by zero
```

Abbildung 13: Fehlermeldung bei Division durch 0 (eigene Darstellung)

Dem könnte durch folgenden Code vorgebeugt werden:

```
def divideNull(self):
    try:
        self.root.ids.ausgabefeld1.text = 3/0
    except:
        self.root.ids.ausgabefeld1.text = 'Division durch 0' \
                                          ' ist nicht erlaubt'
```

An deises Beispiel anlehnend, könnte die Methode *add()* wie folgt erweitert werden, um unabhängig von den Benutzereingaben eine stabile App zu gewährleisten. Die Anpassung ist in Abbildung 14 sichtbar.

```
def add(self):
    try:  # hierdurch werden Eingabefehler abgefangen
        prefix = 'Die Summe lautet: '
        input1 = int(self.root.ids.eingabefeld1.text)
        input2 = int(self.root.ids.eingabefeld2.text)
        summe = input1 + input2
        self.root.ids.ausgabefeld1.text = prefix + \
             str(summe)
    except:  # im Fehlerfall wird dieser Code ausgeführt
        self.root.ids.ausgabefeld1.text = \
            'Eingabefehler!'
```

Abbildung 14: Exception Handling in der Maske

Kommentare im Programm (siehe Codeausschnitt) können auch in dieser Einheit kurz angesprochen und erläutert werden, wofür kein großer Aufklärungsauwand notwendig sein sollte.

6.6.2 Bedingungen, Vergleichsoperatoren, Boolean Datentyp, Gültigkeitsbereich von Variablen

Eine Verfeinerung der Fehlerbehandlung könnte durch Bedingungen realisiert werden, die sich daher als nächstes Konzept im Curriculum eignen würden. Einige Schlagwörter für die Lehrperson zu Bedingungen:

- Überprüfung von Wahrheitskriterien und nur wenn zutreffend, wird die im Codeblock enthaltene Anweisung ausgeführt. Ansonsten Sprung zur nächsten Anweisung;
- Elif optional (Abfrage sonstiger Bedingungen);
- Else am Ende, ebenfalls optional;
- Einführung des Datentyps Boolean;
- Erläuterung von Vergleichsoperatoren. Besonderen Vermerk auf die Abgrenzung zum

Zuweisungsoperator machen, da dies eine häufige Fehlerquelle in Programmen ist;

- Erläuterung logischer Operatoren (and/or, not, etc.)

An verschiedenen einfachen Beispielen sollten die Konzepte vorgeführt werden, z.B.:

```
### Bsp. zu mehreren Bedingungen und >= Operatot
person_alter = 13
if person_alter < 18:
    self.root.ids.ausgabefeld1.text = \
    'Junior'
elif person_alter >= 65:
    self.root.ids.ausgabefeld1.text = \
    'Senior'
else:
    self.root.ids.ausgabefeld1.text = \
    'arbeitspflichtige Population.. :('

### Bsp. zu Boolean Datentyp und 'and' Verknüpfung
person_alter = 24
besitzt_fuehrerschein = True
if person_alter >= 18 and besitzt_fuehrerschein:
    self.root.ids.ausgabefeld1.text = \
    'Die Person darf ain KFZ fahren.'
else:
    self.root.ids.ausgabefeld1.text = \
    'Die Person ist nicht fahrberechtigt.'

### Bsp. zu 'in', Stringvergleich und not
regnerisch = False
tag = 'Freitag'
if tag in ('Samstag', 'Sonntag') and not regnerisch:
    self.root.ids.ausgabefeld1.text = \
    'Es wird ein schöner Wandertag.'
else:
    self.root.ids.ausgabefeld1.text = \
    'Die Berge müssen noch warten...'
```

Weitere Möglichkeiten und Operatoren zu Bedingungen, die in den Ausschnitten ausgelassen wurden, sollten an ähnlichen Beispielen veranschaulicht werden. Die Möglichkeit von Verschachtelungen von Bedingungen sollten auch in Erwägung gezogen werden.

Nach geringfügigen Anpassungen von vorhandenen, sollten Studierende im nächsten Schritt eigene einfache Methoden erstellen, wie Kölling und Rosenberg, (2001) in Ihrer Studie als Richtlinie empfehlen. Als ein einfaches Übungsbeispiel könnte die Eingabemaske um einen Button mit einer Vergleichsmethode erweitert werden, welche die Zahleneingaben vergleicht und das Ergebnis ausgibt, z.B.:

```
def compare(self):

    a = float(self.root.ids.eingabefeld1.text)
    b = float(self.root.ids.eingabefeld2.text)

    vergleichsErgebnis = ''

    if a > b:
        vergleichsErgebnis = 'Die erste Zahl ist größer'
    elif a < b:
        vergleichsErgebnis = 'Die erste Zahl ist kleiner'
    else:
        vergleichsErgebnis = 'Zahlen sind gleich groß'

    self.root.ids.ausgabefeld1.text = vergleichsErgebnis
```

An diesem Beispiel lässt sich auch der Gültigkeitsbereich von Variablen verdeutlichen. Die bisherigen Methodenimplementierungen und die Verwendung von gleichen Variablen in unterschiedlichen Methoden können erwähnt werden (siehe z.B. Code zur Abbildung 11: die Variable *input* wird beispielsweise sowohl in der Methode *enter1()* als auch *enter2()* verwendet, was auf ihre lokale Gültigkeit (auf Methode begrenzt) schließen lässt). Weitere Erklärungen zu Variablengültigkeit innerhalb der Code-Ebenen (Blöcke) und Veranschaulichungen dazu sollten folgen.

Im anschluss kann z.B. die Additionsmethode der Eingabemaske-App als Aufgabe erweitert werden, um genauere Fehlermeldungen zu liefern. Eine Implementierung könnte etwa wie folgt aussehen:

```
def add(self):

    if self.root.ids.eingabefeld1.text == '' and \
      self.root.ids.eingabefeld2.text == '':

        self.root.ids.ausgabefeld1.text = 'Beide Eingabefelder leer!'

    elif self.root.ids.eingabefeld1.text == '':
        self.root.ids.ausgabefeld1.text = 'Eingabefeld 1 leer!'

    elif self.root.ids.eingabefeld2.text == '':
        self.root.ids.ausgabefeld1.text = 'Eingabefeld 2 leer!'

    else:
        try:
            prefix = 'Die Summe lautet: '
            input1 = float(self.root.ids.eingabefeld1.text)
            input2 = float(self.root.ids.eingabefeld2.text)
            summe = input1 + input2
            self.root.ids.ausgabefeld1.text = prefix + str(summe)
        except:
            self.root.ids.ausgabefeld1.text = \
            'Es müssen Zahlen eingegeben werden!'
```

Zusätzliche Aufgaben zu Bedingungen und Vergleichsoperatoren könnten für Einzel- oder Teamübungen vergeben werden.

6.7 Die erste Anwendungsapp – einfacher Währungskonverter

Nachdem sich Studierende mit der Syntax einige Zeit lang auseinandergesetzt haben, Methoden angepasst und auch eigene erstellt haben, wäre die nächste Stufe, einfache Programme von Grund auf zu entwickeln (Kölling & Rosenberg, 2001).

Mehrere Basiskonzepte der Programmierung wurden bereits an generellen Beispielen vorgestellt. Diese sollten nun in einer simplen Anwendungsapp wie z.B. einem Währungskonverter eingesetzt werden, damit deren praktischer Wert verdeutlicht wird. Dadurch soll relativ früh das Gefühl vermittelt werden, dass Studierende das Programmieren bereits im gewissen Maße beherrschen und schon selbst einfache Apps erstellen können.

Die App wird in verschiedenen Zyklen durch Interaktion mit den Studenten aufgebaut, zunächst mit einer ganz einfachen GUI sowie Konverteirungslogik. Anschließend soll die App graduell erweitert werden, bis sie die Form in Abbildung 16 auf Seite 60 erhält. Durch den ganzen Prozess der App-Entwicklung sollen die erlernten Konzepte an einem realitätsnahen Anwendungsbeispiel weiderholt und befestigt werden. Des Weiteren sollen die Grundlagen des Softwareentwicklungsprozesses vorgestellt und die problemlösende Denkweise gefördert werden.

Folgende Schritte ergeben sich für die erste Einheit in diesem Zusammenhang:

1. Anforderungsanalyse:
    - Auseinandersetzung mit dem Begriff des Währungskonverters
    - Umfangseingrenzung
    - Vorerst wird von einem einfachen Szenario ausgegangen: Der Konverter soll mit einem fix vorgegebenen Kurs Euro in Dollar konvertieren:1 € eintspricht 1,13 $.
2. Das Grundgerüst an Code für den Aufruf der App in einem neuen Verzeichnis vorbereiten: Dateien mit einer Unterklasse von App (mit beliebigem, sinngemäßen Namen), sowie deren Instanzierung und Aufruf schreiben:

```
#  Waehrungskonverter.py

from kivy.app import App

class Waehrungskonverter(App):
      pass
```

```
#  main.py

from Waehrungskonverter import *

NeueApp = Waehrungskonverter()

NeueApp.run()
```

Mit diesem Code wird nur eine App erzeugt, die keine implementierungslogik aufweist (*pass* in der Klasse) und auch keine GUI hat. Beim Aufruf sollte nur ein schwarzes Fenster mit der Beschriftung des Klassennamens auftauchen. Dies sollte getestet werden, um ev. Syntaxfehler sofort zu beseitigen, da mit mehr Code der Aufwand zum finden von Fehlerursachen entsprechend steigt. Auf die graduelle Programmerweiterung, vor allem in der Anfangsphase, wenn viele syntaxfehler passieren, sollten Studierende aufmerksam gemacht werden.

3. Visualisierung einer einfachen GUI für die App: eine Zeichnung / Skizze zur Veranschaulichung anfertigen, die ungefähr der Abbildung 15 entspricht
4. Implementierung der GUI mit Erläuterungen:

```
#waehrungskonverter.kv

BoxLayout:
    orientation:'vertical'
    BoxLayout:
        orientation:'vertical'
        Label:
            text: 'Simpler Währungskonverter'
        BoxLayout:
            orientation:'horizontal'
            TextInput:
                id: anzeige_betrag_eingabe
            TextInput:
                text: ' = '
                disabled: True
            TextInput:
                id: anzeige_betrag_ausgabe
                disabled: True
        Label:
            text: 'Kurs: 1 € = 1.13 $'
        Button:
            text: 'konvertieren!'
            on_press: app.konvertieren()
    BoxLayout:
```

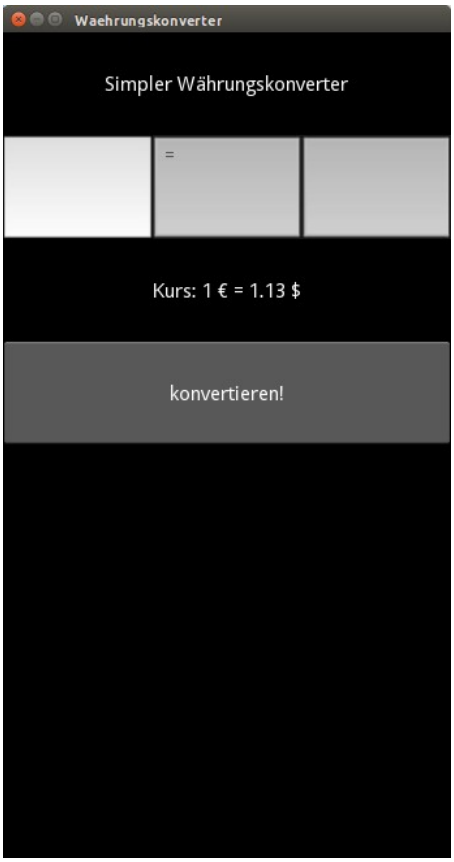


Abbildung 15: GUI eines simplen Währungskonverters

Anmerkungen: Das BoxLayout am Ende wird als Platzhalter verwendet, damit die Textfelder und Button aus Übersichtlichkeitsgründen nicht die gesamte, sondern nur die Hälfte der zur Verfügung stehenden Fläche einnehmen. Die nicht verwendeten Ids und Texteigenschaften werden nicht explizit im Code angegeben. Eingabefelder, die der Benutzer nicht manuell ändern soll, sind deaktiviert (ausgegraut). Es wird eine Methode beim Betätigen des Buttons aufgerufen, die nachträglich implementiert wird, was beim Testen in Betracht gezogen werden muss, da es ansonsten zu Abstürzen der App kommt. Derzeit werden für TextInput Widgets keine Einstellungen für die Textausrichtung unterstützt, daher können die standardmäßigen (oben, links) nicht geändert werden.

Die GUI sollte nicht in einem Zug codiert werden, sondern graduell – von den kleinsten zu den größten Einrückungen im Code – mit Aufrufen zwischendurch, damit die Änderungen verfolgt und nachvollzogen werden können.

5. Im nächsten Schritt soll die Programmlogik mit Studenten diskutiert und aufgebaut werden:
    - Objektorientierte Herangehensweise: der Währungskonverter soll als Klassenkonzept vermittelt werden. Auf Eigenschaften eines solchen Objektmodells sollte eingegangen werden (z.B. Ausgangs-/ Zielwährung und Währungskurs – Einführung der Klassenattribute), sowie auf Funktionen (Methoden). Zunächst wird nur die Methode *konvertieren()* zum Umwandeln von Euro nach Dollar implementiert.
    - Grundsätzlich handelt es sich beim Konvertieren von Währungen um eine einfache Multiplikation mit einem bestimmten Faktor (Kurs der jew. Währung). Das Prinzip soll interaktiv Mit den Studenten an einigen Beispielen mathematisch erläutert werden.
    - Danach wird anhand der Erläuterungen versucht stichwortartig festzulegen, was und in welcher Reihenfolge in die Methode *konvertieren()* eingebaut werden soll, damit das gewünschte Ergebnis geliefert wird. Die Verwendung von **Pseudocode** kann eingeführt werden, durch den die notwendigen Schritte und deren Reihenfolge – also der Algorithmus – festgehalten werden sollen. Eine Basisversion wäre z.B. folgende:

```
input = eingabefeld
betrag_eingabe = zahlenwert(input)
betrag_konvertiert = betrag_eingabe * kurs
output = textwert(betrag_konvertiert)
```

Im Code kann der Pseudocode in Form von Kommentarzeilen eingefügt werden:

```
from kivy.app import App

class Waehrungskonverter(App):
    ausgangswaehrung = '€'
    zielwaehrung = '$'
    kurs = 1.13

    def konvertieren(self):
        #input = eingabefeld
        #betrag_eingabe = zahlenwert(input)
        #betrag_konvertiert = betrag_eingabe * kurs
        #output = textwert(betrag_konvertiert)
```

6. Codierung vor bzw. zusammen mit den Studenten, wie in der Studie von Rubin (2013) empfohlen wird:

```
from kivy.app import App

class Waehrungskonverter(App):
    ausgangswaehrung = '€'
    zielwaehrung = '$'
    kurs = 1.13

    def konvertieren(self):
        #input = eingabefeld
        input = self.root.ids.anzeige_betrag_eingabe.text
        #betrag_eingabe = zahlenwert(input)
        betrag_eingabe = float(input)
        #betrag_konvertiert = betrag_eingabe * kurs
        betrag_konvertiert = betrag_eingabe * self.kurs
        #betrag_ausgabe = textwert(betrag_konvertiert)
        self.root.ids.anzeige_betrag_ausgabe.text = str(betrag_konvertiert)
```

7. Ausführung, Testen: Mit den Studierenden sollen die Testergebnisse und Probleme, auf die gestoßen wird, besprochen und Lösungsvorschläge angeboten werden.

   Mögliche Problemfälle:

   1. Texteingabe;
   2. Dezimalzahleingabe mit Komma als Trennzeichen;

3. Ausgabe hat mehr als 2 Dezimalstellen;
4. überhaupt keine Eingabe.

Interaktiv im Unterricht soll das Abfangen der Problemfälle eines nach dem anderen durchgeführt und getestet werden:

- Problemfall 1 und 2 können vermieden werden durch das Setzen eines Datentypfilters für das TextInput Feld `anzeige_betrag_eingabe` in der .kv Datei: *input_filter: 'float'*.
- Problemfall 3 wird durch das Verwenden der built-in Funktion *round()* behoben, auf die noch im späteren Verlauf des Curriculums genauer eingegangen wird: *betrag_gerundet = round(betrag_konvertiert,2)*.
- Problemfall 5 kann durch eine Bedingunsabfrage vermieden werden.

Als Code sollte sich die Methode konvertieren nach schrittweiser Erweiterung letztendlich wie folgt abbilden lassen:

```
from kivy.app import App

class Waehrungskonverter(App):
    ausgangswaehrung = '€'
    zielwaehrung = '$'
    kurs = 1.13

    def konvertieren(self):

        if self.root.ids.anzeige_betrag_eingabe.text =='':
            return

        input = self.root.ids.anzeige_betrag_eingabe.text

        betrag_eingabe = float(input)
        betrag_konvertiert = betrag_eingabe * self.kurs
        betrag_konv_gerundet = round(betrag_konvertiert,2)

        self.root.ids.anzeige_betrag_ausgabe.text = str(betrag_konv_gerundet)
```

8. Im letzten Schritt soll der Code zum Smartphone übertragen und im Kivy Launcher erneut ausgeführt und getestet werden
9. Als Aufgabe könnten die Studenten angeregt werden, die visuelle Darstellung anspruchsvoller zu gestalten sowie Erweiterungsansätze für die App zu überlegen. Eventuell können auch erste Implementierungsversuche diesbezüglich unternommen werden, die in der nächsten Einheit analysiert und besprochen werden können.

6.8 Währungskonverter Erweiterung/Verfeinerung

In der vorigen Einheit wurde eine simple Wärungskonverter-App entwickelt, die jedoch sehr begrenzt im Funktionsumfang ist. Mit der App können Beträge nur von Euro nach Dollar konvertiert werden und zwar zu einem fest (hardcodiert) vorgegebenen Wechselkurs. Ein Währunskonverter sollte zumindest noch dem Benutzer ermöglichen, den Wechselkurs zu ändern oder auch die Beträge in eine andere Währung als Dollar zu konvertieren. Daher lässt sich die App um noch einige Funktionalitäten erweitern, z.B.:

- Festlegung der Ausgangs-/Zielwährung
- Festsetzung des Wechselkurses
- Ausgabe der meistverwendeten Beträge bei der Konvertierung
- Ausgangs- und Zielwährung vertauschen
- ev. andere Funktionen

Des Weiteren könnten zusätzliche Widgets bzw. deren Eigenschaften verwendet werden, um die Benutzerfreundlichkeit der App zu verbessern.

Eine Diskussion bez. Erweiterungsvorschlägen und -möglichkeiten sollte im Unterricht erfolgen und eventuell einige Implementierungsversuche aus der Hausübung besprochen werden.

In dieser Einheit sollte ein gemeinsamer Entwurf der erweiterten Währungskonverter-App erarbeitet und implementiert werden. Eine mögliche Lösung, zunächst für die GUI, wäre (in Endausführung) folgende(Abbildung 16):

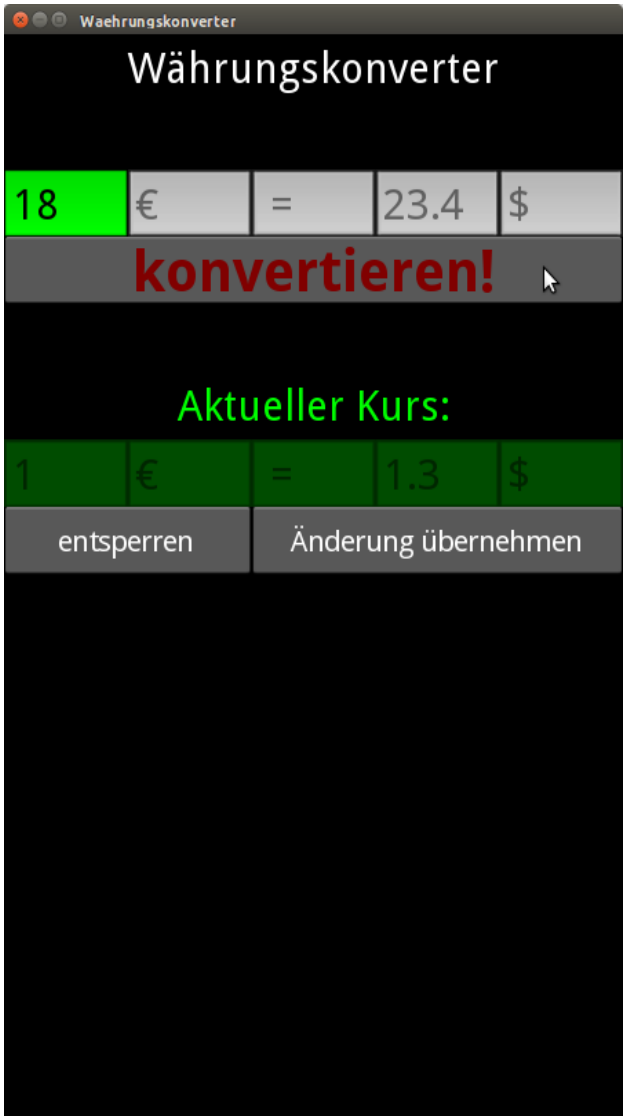


Abbildung 16: Währungskonverter GUI, erweitert

Die GUI soll auf der vorhandenen aufbauen und schrittweise vor den Studenten erweitert werden, um die erlernten Konzepte zu wiederholen und den Entwicklungsprozess vorzuzeigen. Der gesamte Code dazu kann dem Anhang 8.2.2 entnommen werden.

Anmerkungen: Das Eingabefeld wird mit einem Hint versehen und hellgrün hervorgehoben. Beim Fokus wird der Hint-Text automatisch gelöscht. Im obigen Teil der GUI ist nach der

Überschrift ein Label als Platzhalter definiert, damit die Felder, zwecks Übersichtlichkeit, nicht zu hoch in der GUI sind. Die Felder sind bis auf das eine Eingabefeld gesperrt, da sie statisch sind und vom User nicht geändert werden sollen. Die Ausgangs-/Zielwährung, sowie der aktuelle Kurs sind in der neuen Maske nicht mehr im Label enthalten, sondern in TextInput Feldern (mit grünem Hintergrund) vorgegeben, da sie änderbar sein sollen. Sie zeigen in dieser Maske die Klassenattribute an und sind standardmäßig gesperrt, damit keine versehentlichen Änderungen gemacht werden können. Durch einen Buttonklick können sie jedoch für Änderungen freigegeben werden.

Nachdem man sich auf eine Benutzeroberfläche geeinigt hat, und diese auch durch mehrere Zyklen und Tests implementiert wurde, sollte die Implementierung der Programmlogik besprochen werden. Die *konvertieren()* Methode kann (zumindest vorerst) ohne Änderungen aus der ursprünglichen App übernommen werden. Des Weiteren müssen die Methoden *kurs_entsperren()* und *kurs_aendern()* implementiert werden. Im Unterricht sollte bezüglich der Umsetzung eine Diskussion erfolgen.

Zunächst wird die einfachste Änderung durchgeführt, was in diesem Fall die Implementierung der Methode *kurs_entsperren()* ist. Dabei sollen die drei relevanten Felder des Kurses einfach zur Eingabe durch folgenden Code freigegeben werden:

```
def kurs_entsperren(self):
      self.root.ids.aktuell_ausgangswaehrung.disabled = False
      self.root.ids.aktuell_kurs.disabled = False
      self.root.ids.aktuell_zielwaehrung.disabled = False
```

Wenn gewünschte Anpassungen in den jeweiligen Feldern erfolgt sind, sollen diese durch das Betätigen des entsprechenden Buttons übernommen werden. Das bedeutet, dass die Klassenattribute geändert werden und dies im Nachhinein auch in der GUI ersichtlich werden soll:

```
def kurs_aendern(self):

      self.ausgangswaehrung = self.root.ids.aktuell_ausgangswaehrung
      self.kurs = float(self.root.ids.aktuell_kurs.text)
      self.zielwaehrung = self.root.ids.aktuell_zielwaehrung.text

      self.anzeige_aktualisieren()
```

Im Code wird eine noch nicht definierte Methode *anzeige_aktualisieren()* aufgerufen. Es sollte angemerkt werden, dass auch 'interne' (also nicht unbedingt mit einem Button verbundene) Methoden existieren können, die als Hilfsmethoden zur besseren Übersichtlichkeit / Wiederholungsvermeidung im Code dienen. Die Methode soll gleich im Anschluss implementiert werden:

```
def anzeige_aktualisieren(self):
      self.root.ids.anzeige_ausgangswaehrung.text = self.ausgangswaehrung
      self.root.ids.anzeige_zielwaehrung.text = self.zielwaehrung
      self.root.ids.aktuell_ausgangswaehrung.disabled = True
      self.root.ids.aktuell_kurs.disabled = True
      self.root.ids.aktuell_zielwaehrung.disabled = True

      if self.root.ids.anzeige_betrag_eingabe.text != '':
            self.konvertieren()
```

Durch die Methode werden die Klassenattribute des Währungskonverters (Ausgangs-, Zielwährung, Kurs) in der Eingabezeile neu gesetzt, die Felder des aktuellen Kurses werden wieder für unerwünschte Änderugnen deaktiviert und die Konvertierungsmethode wird mit den neuen Werten erneut angestoßen.

Die entworfene App sollte nun getestet und ev. Fehlerfälle besprochen werden. Das Testing könnte generell auch als eine Herasuforderung für die Studenten dargestellt werden und es könnten bestimmte Belohnungen für die Person ausgesetzt werden, die die meisten Fehler entdeckt.

Im vorhandenen Code sind durch den *input_filter* und die Deaktivierung der nicht relevanten Input Felder bereits viele Prolemquellen abgefangen worden. Zumindest zwei Problemfälle dürften aber noch beim Testen auffallen:

- wenn eines der Felder vom aktuellen Kurs leer ist
- wenn die Ausgangs- und Zielwährung gleich sind

Durch folgende Ergänzungen in den Methoden könnten diese verhindert werden:

```
def kurs_aendern(self):
      if self.root.ids.aktuell_ausgangswaehrung.text == '' or \
            self.root.ids.aktuell_kurs.text == '' or\
            self.root.ids.aktuell_zielwaehrung.text == '' or\
            self.root.ids.aktuell_ausgangswaehrung.text == \
            self.root.ids.aktuell_zielwaehrung.text:

            self.root.ids.kurslabel.text = 'Fehler in Kurseingabe!'
            self.root.ids.anzeige_betrag_eingabe.disabled = True
            return
      else:
            #...

def anzeige_aktualisieren(self):
      #...
      if self.root.ids.anzeige_betrag_eingabe.disabled == True:
                  self.root.ids.anzeige_betrag_eingabe.disabled = False
      if self.root.ids.kurslabel.text == 'Fehler in Kurseingabe!':
            self.root.ids.kurslabel.text = 'Aktueller Kurs:'
```

Im Anschluss empfiehlt es sich, den entwickelten und getesteten Code zum Smartphone zu übertragen, damit Studierende ihn als ihr erstes abgeschlossenes Anwendungsapp-Projekt betrachten können, das die sie gemeinsam im Unterricht erstellt haben.

Das Ziel dieser Einheit ist es, vor den Studenten und durch Interaktion mit ihnen zu Programmieren und neben dem praktischen Anwenden von Programmierkonzepten den Software-Entwicklungsprozess auf dem Basislevel zu zeigen.

Als (Fleiß)Aufgaben könnte z.B. die Implementierung einer *umschalten()* Methode sein, welche die Ausgangs- und Zielwährung vertauscht sowie den Wechselkurs entsprechend anpasst. Des Weiteren könnte eine ähnliche App erstellt werden für die Konvertierung von km nach Meilen, kg nach Pfund und Celsius nach Fahrenheit (und umgekehrt), welche für einen Auslandsaufenthalt nützlich sein kann.

6.9 Datenstrukturen, Funktionen

Nachdem eine erste funktionale App entwickelt und zum Smartphone übertragen wurde, sollte das Interesse der Studierenden zusätzlich geweckt worden sein. Die Möglichkeit, relativ einfach eigene Apps erstellen zu können, dürfte lt. bereits erwähnten Studien und auch aus eigener Erfahrung sehr positive Auswirkungen auf die Motivation zum weiteren Ausbau der Programmierkenntnisse und Fortschritt diesbezüglich haben. Einige zusätzliche Apps, die Studierende im Rahmen der Hausübung erhalten haben könnten, sollte sie selbstsicherer im Umgang mit Code und dem Softwareentwicklungsprozess gemacht haben.

Im Folgenden sollten die Kenntnisse um noch einige wichtige Konzepte des Programmierens erweitert werden, damit Studierende auch für etwas komplexere Anforderungen, die später in weiteren Apps verdeutlicht werden, gerüstet sind.

6.9.1 Listen, Tupel

An dieser Stelle bietet sich z.B. die Vorstellung weiterer Datenstrukturen an, die oft in Programmen verwendet werden, worunter z.B. Listen und Tupel fallen würden. Im Folgenden sind stichwortartig die Erläuterungen der Begriffe und Beispiele der Verwendung angeführt.

1. Listen
    - Sammlung mehrerer Datensätze, die auch vom unterschiedlichen Datentyp sein können;

      ```
      eine_liste = [3, 'text1', 7, 'e' ]
      ```

    - änderbar: Zugriff durch indexing/slicing (wie string);
    - viele Methoden für den Umgang mit Listen, z.B.:

```
eine_liste.append('zusatzitem') # eine_liste = [3, 'text1', 7, 'e', 'zusatzitem' ]

eine_liste.count('e') # 1

len(eine_liste) # 5

eine_liste [5:7] = ['a', 'b'] #eine_liste=[3, 'text1', 7, 'e', 'zusatzitem', 'a', 'b' ]

eine_liste.index('a') # 5

del eine_liste[4:6] # eine_liste = [3, 'text1', 7, 'e', 'b' ]

eine_liste.sort() # Liste sortieren

eine_liste.reverse() # Reihenfolge umkehren

eine_liste.min() # kleinstes Element ausgeben

#...
```

2. Tupel

- Sammlung mehrerer Datensätze, die auch vom unterschiedlichen Datentyp sein können (wie Listen);
- Wesentlicher Unterschied: Tupel sind **nicht** änderbar, daher nur bei solchen Anforderungen im Programm zu verwenden;
- weniger Methoden verfügbar, dafür effizienter.

```
tup = ("a", 2, "33")

tup[1] = 7 # nicht möglich → Fehlermeldung

len(tup) # 3

tup[1] # 2

max(tup) / min(tup) # Ausgabe des Maximums/Minimums

#…
```

Als Anwendungsfall wäre etwa eine simple Einkaufs-App geeignet, die in ungefähr wie in Abbildung 17 ausschauen könnte:

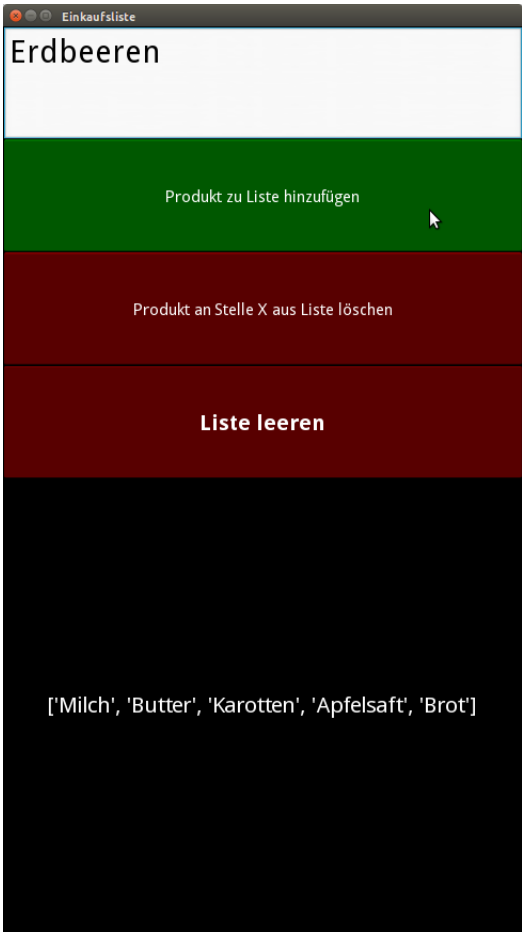


Abbildung 17: Simple Einkaufsliste - App Übung zu Listen

In dieser App können Produkte in eine Liste gespeichert werden, einzelne Produkte können anhand ihrer Position in der Liste ermittelt und gelöscht werden. Das Löschen der gesamten Liste nach einem Einkauf ist ebenfalls vorgesehen. Änderungen und Erweiterungen oder auch die Implementierung von anderen Apps zur Übung von Listen seitens der Lehrperson sind natürlich willkommen. Es sollte insbesondere auf die Indizierung Acht gelegt werden (Beginn bei 0, sowie 'index out of range' Problematik).

Der gesamte Code zu der App kann Anhang 8.2.3 entnommen werden.

6.9.2 Funktionen, Argumente, Rückgabewert

Funktionen sind ein weiterer wichtiger Begriff für den Programmiereinstieg, der jedoch durch den objektorientierten Ansatz und die Verwendung von Methoden konzeptuell schon zu Beginn vorgestellt wurde. Einige zusätzliche Merkmale und Einsatzmöglichkeiten, die oft zur Anwendung kommen, sollten aber noch theoretisch erläutert und praktisch an Beispielen vorgezeigt werden. Unter anderem sollte auf folgende Eigenschaften von Methoden hingewiesen werden:

- Code-Blöcke, welche auf die Erledigung ganz bestimmter Teilaufgaben des Gesamtprogramms ausgerichtet sind.
- Methoden werden im Code aufgerufen. Der Aufruf aus der GUI muss nicht unbedingt gegeben sein - es können auch 'interne' Methoden erstellt werden, die in anderen verwendet werden können (Verweis auf die Methode *anzeige_aktualisieren()* in Kapitel 6.8 zur Erweiterung des Währungskonverters).
- Weise, Code zu organisieren: Vermeidung von redundanten Anweisungsblöcken und übersichtlichere Gestaltung von Code.
- Führen bestimmte Aufgaben aus und können ein Ergebnis zurückliefern, mit dem weiter im Programm vorgegangen wird.
- Können Argumente als Input Empfangen, anhand dessen die Abarbeitung und der Output unterschiedlich ausfallen kann.
- Aufruf mit Klammern, Argumente (optional) werden innerhalb der Klammern übergeben.

```
def summe(self,zahl1,zahl2):
      summe = zahl1 + zahl2
      return summe

def umfang_rechteck(self,laenge,breite):
      umfang = (laenge + breite) * 2
      return umfang

def umfang_rechteck(self,laenge,breite):
      return = (laenge + breite) * 2


def umfang_rechteck(self,laenge,breite):
      umfang = self.summe(laenge,breite) * 2 #werden Variable zugewiesen
      return umfang
```

Tipp für Studierende bei der Verwendung von Methoden: self bei Definition **in** der Klammer, bei Verwendung **vor** der Klammer (und der Methode)

Zur Verdeutlichung von Funktionen könnte beispielsweise eine App zur Berechnung des BMI (Body Mass Index) sowie BFP (Body Fat Percentage) herangezogen werden, um neben Programmierkenntnissen auch das Gesundheitsbewusstsein zu fördern. Theoretische Grundlagen zu den Begriffen und deren Berechnungslogik können im Internet nachgeschlagen werden[23]. Grundsätzlich handelt es sich dabei um die Feststellung anhand bestimmter Eingangsgrößen, ob eine Person über-/untergewichtig ist oder normales Gewicht aufweist, sowie die Berechnung des Fettprozentanteils am Körper. Eine graphische Umsetzung könnte etwa wie in Abbildung 18 implementiert werden (Code dazu in Anhang 8.2.4):

Abbildung 18: BMI Berechnungsapp. Übung zu Funktionen

23 http://www.bmi-rechner.net/
https://en.wikipedia.org/wiki/Body_mass_index
https://en.wikipedia.org/wiki/Body_fat_percentage

Um die Verwendung von Funktionen mit Argumenten und Rückgabewerten zu verdeutlichen, sollten im Klassen-Code zunächst die Berechnungsmethoden der App implementiert werden.

```
from kivy.app import App
class BMICalculator(App):

      def bmi_berechung(self,kg,cm):
            bmi = kg / ((cm/100)**2)
            return bmi
```

Danach sollte die Implementierung der Buttonmethode mit dem Aufruf der BMI Berechnung erfolgen, um die Verwendung des Rückgabewerts zu veranschaulichen (der Rückgabewert der Funktion *bmi_berechnung* wird der Variable *my_bmi* zugewiesen, die weiter im Programm verwendet wird):

```
    def werte_ermitteln(self):
          groesse = float(self.root.ids.groesse_cm.text)
          gewicht = float(self.root.ids.gewicht_kg.text)

          my_bmi = self.bmi_berechung(gewicht,groesse)
          self.root.ids.bmi.text = str(my_bmi)
```

Für eine visuelle Verdeutlichung der BMI-Werte könnte eine entsprechende Färbung des Ausgabefeldes als praktische Übung erfolgen, z.B. gelb für untergewichtig, grün für durchschnittlich und rot für übergewichtig. Dadurch wird noch eine Funktion mit Argumenten definiert, Bedingungen und logische Verknüpfungen wiederholt, sowie Rückgabewerte und Tupel (für die Farbdefinition) befestigt.

```
def kolorieren(self,wert):
      if wert < 18.5:
            return (1,1,0,1)
      elif wert >=18.5 and wert < 25:
            return (0,1,0,1)
      else:
            return (1,0,0,1)
```

Danach muss die *werte_ermitteln* Methode auch entsprechend angepasst werden, was wiederum die graduelle Erweiterung eines Programms verdeutlicht. In Folge sollten im Rahmen von Übungen weitere Berechnungsfunktionen implementiert werden: Ausgabe des Körperfettanteils in Prozent anhand des BMI, sowie Berechnung des Idealgewichts für die eingegebenen Daten. Der Codevorschlag zur gesamten App befindet sich in Anhang 8.2.4.

6.10 Schleifen

Im nächsten Schritt bieten sich Schleifen an, als eines der wichtigsten Programmierkonzepte eingeführt zu werden. Dieses Konzept erscheint womöglich nicht so intuitiv wie viele andere, mit denen man im weiten Sinne auch sonst wahrscheinlich konfrontiert gewesen ist (Variablen, Eigenschaften von Feldern, Datentypen, etc.), deswegen werden Schleifen etwas später im Curriculum eingeführt und an möglichst einfachen Anfangsbeispielen verdeutlicht.

Die Unterschiede und mögliche Ausprägungen sollen an simplem Mustern erläutert werden. Es kann eine simple Eingabemaske wie in Kapitel 6.2.1 oder 6.5 herangezogen werden, die durch den Buttonklick iterative Ausgaben bewirkt, z.B.:

```
def schleifenausgabe(self):

      ausgabe = ''

      for i in range (1,11):
            ausgabe = ausgabe + 'aktueller wert von i ist: ' + str(i) + '\n'

      self.root.ids.ausgabefeld1.text = ausgabe
```

Weitere Beispiele bezüglich der Ausprägungen von Schleifen, z.B.:

```
#Schrittweite bei der Iteration
for i in range (1,11,2):
      ausgabe = ausgabe + 'aktueller wert von i ist: ' + str(i) + '\n'

#Verwendung von Variablen auch möglich
 biswert = 13
 for z in range (1,biswert,3):
      ausgabe = ausgabe + 'aktueller wert von i ist: ' + str(i) + '\n'
```

```
#Verwendung mit Listen
korbliste = ['Apfel','Birne','Banane','Mango']

for element in korbliste:
      ausgabe = ausgabe + element + '\n'

#Methode zum automatischen befüllen einer Liste mit Zahlen
def zahlenliste_erstellen(self, bis_zahl):

       a = []
       for x in range(1, bis_zahl):
            a.append(x)



#Erläuterung der While-Schleife:
def countdown(self):
      c = 10
      ausgabe = ''
      while (c>0):
            ausgabe = ausgabe + str(c)
            c = c – 1
      self.root.ids.ausgabefeld1.text = ausgabe
```

Als Anwendungsfälle für die praktische Übung könnten bereits erstellte Apps um einige Features erweitert werden. Beispielsweise könnten in der Währungskonverter-App aus Kapitel 6.8 häufig verwendete Beträge beim Konvertieren durch Iterationen angezeigt werden(Abbildung 19, Seite 72):

```
def standardbetraege_anzeigen(self):

      spalte10 = ''
      Spalte50 = ''

      for i in range(10,101,10):
            spalte10 =  spalte10 +\
            str(i)  + self.ausgangswaehrung + ' = ' +\
            str(i * self.kurs) + ' '+ self.zielwaehrung + '\n'

      for i in range(50,501,50):
            spalte50 =  spalte50 +\
            str(i)  + self.ausgangswaehrung + ' = ' +\
            str(i * self.kurs) + ' '+ self.zielwaehrung+ '\n'

      self.root.ids.betraege10er.text = spalte10
      self.root.ids.betraege50er.text = spalte50
```

Abbildung 19: Währungskonverter App mit Ausgabe von Standardbeträgen

Des Weiteren könnte die Einkaufs-App aus Kapitel 6.9.1 um die Prüfung durch eine For-Schleife erweitert werden, ob ein Bestimmtes Element sich bereits in der Liste befindet und es ggf. nicht erneut anlegen. Dabei wird auch ein Beispiel der Verschachtelung von Konzepten (Bedingungen innerhalb der Schleife) dargeboten, sowie Funktionen mit Argumenten und Rückgabewerten wiederholt:

```
def vorhanden(self,element):
      for x in self.einkaufskorb:
            if element == x:
                  return True
            else:
                  return False

def enter_item(self):
      new_item = self.root.ids.eingabefeld1.text
      if not self.vorhanden(new_item):
            self.einkaufskorb.append(new_item)
            self.root.ids.ausgabefeld1.text = str(self.einkaufskorb)
      self.root.ids.eingabefeld1.text = ''
```

6.11 Verwendung von Modulen und built-in Funktionalitäten

Im bisherigen Verlauf des Curriculums wurden schon viele Konzepte vorgestellt, die bereits das Erstellen von einfachen funktionellen Apps ermöglichen. Um die große Flexibilität beim Programmieren und die vielen Werkzeuge in Form von vorgefertigten Code für die eigenen Apps als Anreiz in Aussicht zu stellen, sollte ansatzweise der Umgang mit built-in Funktionen und Modulen (Standardbibliothek) angesprochen werden.

Die built-in Funktionen sind vorgefertigte Anweisungen, die bestimmte Aufgaben erfüllen und in die Python Sprache inkorporiert sind. Einige davon wurden bereits in den vorigen Kapiteln vorgestellt, wie z.B. *int()*, *str(), float()*, *round()*. An dieser Stelle sollen sie nochmal theoretisch erläutert werden.[24] Einige davon sollten an Beispielen verdeutlicht werden, z.B.:

- Es können noch weitere String Manipulationen als die bisher verwendeten herangezogen werden.[25] Beispiele wären str.capitalize(), str.replace(), str.count(), Vergleiche (>, <), str.sort() etc.
- round(zahl[,dezimalstellen]): Runden einer Zahl auf eine optionale Anzahl von Dezimalstellen. Wenn keine Dezimalstellen angegeben werden, wird auf die nächste ganze Zahl gerundet (diese Funktionalität wurde schon mehrmals im Code verwerdet).
- min(arg1,arg2,arg-n), max(arg1,arg2,arg-n): Liefert das kleinste bzw größte Element aus den Klammern.
- len(s): Ausgabe der Länge eines Objektes – z.B. Liste, String, etc.
- range(start, stop[,step]): Ausgabe eines Sequenzbereiches - range(0, 10, 3) → [0, 3, 6, 9]
- eval(x): evaluiert den Ausdruck in der Klammer, z.B.:
  - x = 3 eval('x+4') ergibt 7
  - eval(10+3*2/(9-7)) ergibt 13

  Mit dieser built-in Funktion können Studierende bereits das Programmieren eines einfachen Taschenrechners mit entsprechender GUI als Aufgabe erhalten, mit beispielsweise folgender Ausführung(Abbildung 20):

24 Eine ausführliche Liste von built-in Funktionen ist unter https://docs.python.org/2/library/functions.html zu finden

25 Siehe auch https://docs.python.org/2/library/stdtypes.html#string-methods

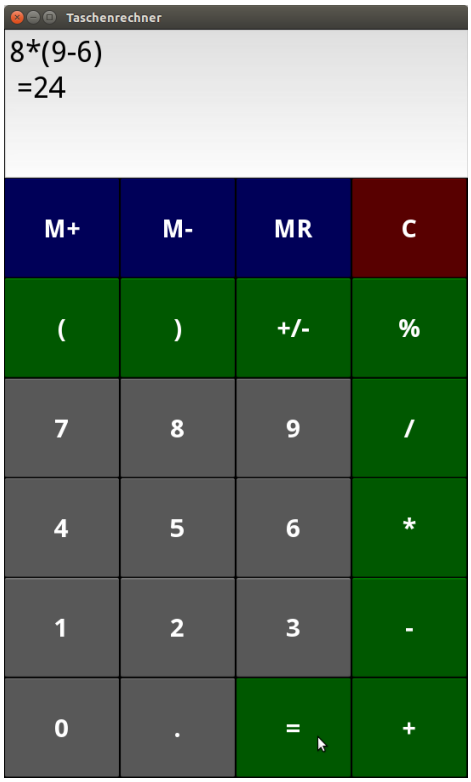


Abbildung 20: App: Simpler Taschenrechner

Eine der Anforderungen wäre, im Display mit dem Ergebnis auch den eingegebene Ausdruck angezuzeigen. Das Ergebnis soll als Anfangsinput für eventuelle Folgeberechnungen dienen. Die Umsetzung einer Methode für Buttonklicks könnte etwa wie folgt ausschauen:

```
def pressed(self,char):

      anzeige_aktuell = self.root.ids.display.text
      anzeige_neu = ''
      if '=' in anzeige_aktuell:
            ergebnis_index = anzeige_aktuell.find('=')+1
            anzeige_neu = str(anzeige_aktuell[ergebnis_index:]) + char
      else:
            anzeige_neu = anzeige_aktuell + char

      self.root.ids.display.text = anzeige_neu
```

Durch die Methode *pressed()* werden die Anwendung der built-in Funktionen eval() str.find() vorgeführt, sowie das String-Slicing erneut aufgegriffen. Der gesamte Code zum Lösungsvorschlag der Taschenrechner – App berfindet sich im Angang 8.2.5.

Im Zweiten Teil der Einheit können, anlehnend an die built-in Funktionen, Module vorgestellt werden.

Module sind im Grunde genommen nichts anderes als bereits vorhandener Code, enthalten und entsprechend organisiert in Textdateien. Um dessen Funktionalitäten verwenden zu können, muss dieser in das aktuelle Programm durch eine sog. Import-Anweisung zur Verfügung gestellt werden. Der Unterschied zu den built-in Funktionen ist, dass Module nicht direkt in die Python-Sprache eingebaut sind, sondern, ähnlich zu Plug-ins, an den eigenen Code 'hinzugefügt' werden müssen.[26]

Zur Demonstration der Unterschiede zwischen Modul und Built-in Funktion könnte eine Modulfunktion verwendet werden, ohne dass das jeweilige Modul importiert wurde. Ein gutes Beispiel wäre, das Importstatement, das in jeder App verwendet wird, auszulassen, um den Effekt zu verdeutlichen – Abbruch beim Aufruf der App mit aussagekräftiger Fehlermeldung (Abbildung 21).

```
    class Tictactoe(App):
NameError: name 'App' is not defined
```

Abbildung 21: Fehlermeldung bei fehlendem Import

Die Importarten sollten erläutert und an Beispielen verdeutlicht werden:

- import modulname
- from modulname import funktionsname
- from modulname import *

Beispiele:

Modul random, das mit Zufälligkeit verbundene Operationen ausführt:

- random.choice((1,2,3,4,5,6,7,8,9,10)): irgendeine Sequenz(String auch möglich)
    - Auswahl von einem der angegebenen Elemente
- random.randint(1,10): von-bis inkludiert
    - Auswahl aus dem angegebenen Zahlenbereich

26 Die Sammlung aller mit Python mitgelieferten Module (Standardbibliothek) kann unter https://docs.python.org/2/library/ aufgerufen werden

- random.shuffle(sequenz)
    - Reihenfolge der Elemente in Sequenz willkürlich anordnen

Folgende Aufgaben könnten sich in dem Zusammenhang ergeben:

- Aufgabe: Kopf oder Zahl randomisiert ausgeben und die Anzahl der jeweiligen Treffer ermitteln
- Eine Aufgabe könnte in etwa sein, Zufallszahlen zu generieren, die in das RGBA Format zur Bestimmung der Farbe eines Buttons eingesetzt werden. So könnte nach jedem Button-Klick dessen Farbe geändert werden.
- Eine Randomisierte Zahl generieren und diese den Benutzer (mit Zählung der Versuche) erraten lassen, durch Hinweise ob sie größer oder kleiner als die von ihm eingegebene (geratene) Zahl ist.

Das Modul String hat relativ viele Funktionen, die zum Übungszwecken herangezogen werden könnten: string.uppercase, string.lowercase, etc.

Ein weiteres Modul, für Aufgaben geeignet, wäre datetime, eine Bibliothek mit ausführlichen Datums- und Zeitfunktionalitäten sowie diversen Formatierungsmöglichkeiten. Unter anderem könnten Studierende aufgefordert werden, ihr Alter oder Sternzeichen anhand des Geburtsdatums zu ermitteln, ihr Alter in Tagen anzugeben, etc.

Es sollten noch die wichtigsten theoretischen Erläuterungen zu Modulen stattfinden:

- importiert werden kann aus der Python library oder auch aus eigenem Code
- beim Import werden alle eventuellen 'Untermodule' automatisch importiert (wenn z.B. das zu importierende Modul selbst andere Modulimporte enthält)
- Erwähnung der Hintergründe von Modulen und deren Vorteile
    - Verwendung von vielen Funktionalitäten, die man nicht selbst programmieren muss
    - Wiederverwendbarkeit von Code (auch eigenem)
    - Vermeidung von Redundanz
    - Übersichtlichkeit, kein copy-paste von Code
    - Erleichterte Wartung

6.12 Wiederholung und Vertiefung der Konzepte an einer Spiel-App (1)

Leutenegger und Edgington (2007) schlagen vor, für die Einführung ins Programmieren den Ansatz vor, Spiele zu entwerfen. Es wird darauf verwiesen, dass die Mehrheit von Personen unter 30 gelegentlich bis oft auf Computern spielt, und viele sind sogar mit Computerspielen als ständigen Begleitern aufgewachsen. Dies deutet auf großes Interesse dieser Zielgruppe an Spielen hin. Die Autoren zeigten in der Studie, dass dies als eine sehr motivierende Wirkung für die Einführung ins Programmieren verwendet werden kann. Spiele sind mit Spaß verbunden und Spaß ist ein guter Grund sich mit einer Tätigkeit länger auseinanderzusetzen. Das Interesse wir dabei vetieft und bleibt länger erhalten Man ist eher dazu bereit, sich etwaigen Schwierigkeiten zu stellen und geduldiger Lösungswege zu suchen.

Andere argumentieren, dass Studenten sich mit Spielprogrammierung viel mehr bemüht und mehr gelernt haben, als es mit herkömmlichen Programmieraufgaben der Fall gewesen wäre. Oft wurden die Erwartungen auch übertroffen (Becker, 2001). Die These, dass die Programmierung von Spielen eine positive Auswirkung auf den Lerneffekt hat, stärken auch weitere Studien (Argent et al., 2006; Lorenzen & Heilman, 2002; Sweedyk, deLaet, Slattery, & Kuffner, 2005)

Darauf aufbauend sollten in einem Curriculum, neben der Motivation durch Verwendung neuer Technologien, auch die Motivation durch das sogenannte Game Based Programming geschöpft werden, um möglichst gute Resultate bei Programmieranfängern zu erzielen. Es sollen daher auch einfache Spiele programmiert werden, die eine Interaktion und visuelles Feedback in Echtzeit beinhalten, da sie anregend auf die Lernperson wirken. Dabei sollten bekannte Spiele herangezogen werden, damit die Regeln und Abfolge (Algorithmus) bekannt sind und man sich nur auf die Implementierung konzentrieren kann und nicht auf zusätzliches Erlernen eines neuen Spieles. Die Umsetzung darf dabei natürlich nicht überfordernd sein, damit keine kontraproduktiven Auswirkungen ausgelöst werden.

Ein Beispiel dafür wäre das bekannte Spiel **Tic Tac Toe**. Es hat eine relativ einfache Logik und ist graphisch auch nicht zu anspruchsvoll. Es kann mit bereits vorhandenem Wissen an graphischen Elementen umgesetzt werden, ohne auf die relativ komplizierte Canvas Klassenpalette für fortgeschrittene Grafik zugreifen zu müssen.

Im folgenden wird eine schrittweise Entwicklung vorgeschlagen, die im Unterricht interaktiv erfolgen sollte.

Im ersten Schritt erfolgt eine Anforderungsanalyse für eine Basisversion des Spieles. Eine einfache GUI mit 9 Schaltflächen für das Spiel, zwei Buttons für die Auswahl welcher Spieler beginnen soll, sowie ein Button für ein neues Spiel sollten dabei als Ergebnis für die Benutzerschnittstelle herauskommen. Die Implementierung könnte wie in Abbildung 22 ausschauen:

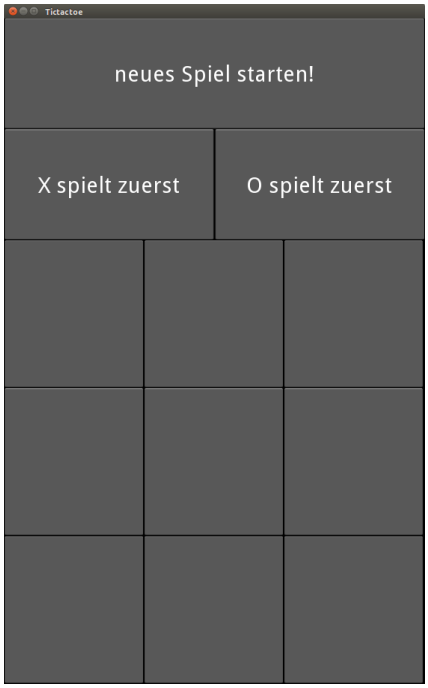


Abbildung 22: Tic-tac-toe App - Entwurf 1

Im Folgeschritt soll die Programmlogik implementiert werden. Studierende sollen aufgefordert werden, eine einfache Beschreibung des Spielverlaufs und der Spielregeln in eigenen Worten zu fassen und eine stichwortartige Anleitung zu schreiben. Dadurch soll das Zerlegen von Aufgaben und die algorithmische Denkweise geübt werden, in etwa wie folgt:

- Spieler der beginnen soll (entweder X oder O) wird festgelegt;
- Der Spieler, der an der Reihe ist, wählt ein Feld aus. Diese wird mit dem Zeichen des jeweiligen Spielers beschriftet (X oder O);
- nach jedem Zug ist der andere Spieler an der Reihe;
- der Spieler, der als erster drei zusammenhängende Felder (entweder senkrecht, waagerecht oder quer) mit seinem Zeichen beschriftet, hat gewonnen.

Des Weiteren sollte ein Gedankenaustausch über die Umsetzung im Code erfolgen.

- Was wird der Standardablauf bei der Verwendung dieser App sein?
- Was soll passieren wenn ein Feld im Spiel gedrückt wird?
- Was soll passieren wenn die Spielerauswahl getroffen wird?
    - Was soll passieren, wenn das Spiel ohne Reihenfolgewahl begonnen wird?
- Was soll der 'neues Spiel starten' Button bewirken?

Eine entsprechende Darstellung in Pseudocode sollte mit den Studenten erarbeitet werden, z.B.:

```
Spiel TicTacToe:
      aktueller_spieler = X #standardmäßig startet X

      button_gedrueckt(button_x):
            button_x.text = aktueller_spieler
            spieler_wechseln()

      spieler_wechseln():
            wenn  aktueller_spieler = X
                  dann setze  aktueller_spieler = O
            sonst setze aktueller_spieler = X

      setze_spieler(zeichen):
            aktueller_spieler = zeichen

      neues_spiel():
            alle button.text = ''
```

Die Logik zur Feststellung des Gewinners soll zu diesem Zeitpunkt noch außer Acht gelassen werden. Der Code soll im Unterricht mit aktiver Beteiligung und Anmerkungen seitens der Studierenden implementiert werden. Eine mögliche Ausprägung könnte so ausschauen:

```
from kivy.app import App

class Tictactoe(App):

	aktueller_spieler = 'X' #standardmäßig startet X

	def button_gedrueckt(self,id):
		self.root.ids[id].text = self.aktueller_spieler
		self.spieler_wechseln()

	def spieler_wechseln(self):
		if self.aktueller_spieler == 'X':
			self.aktueller_spieler = 'O'
		else:
			self.aktueller_spieler = 'X'

	def neues_spiel(self):
		for i in range(0,9):
			b = 'btn' + str(i)
			self.root.ids[b].text = ''

	def setze_spieler(self, zeichen):
		self.aktueller_spieler = zeichen
```

Im Folgenden sollte eine Evaluierung der App sowie Tests erfolgen und Verbesserungs- / Erweiterungsideen eruiert werden. Ein allgemeines Gefühl für den gesamten Softwareentwicklungsprozess und dessen Phasen sollte vermittelt werden.

Relativ bald dürfte als Mangel festgestellt werden, dass bei jedem klick (auch bei bereits befüllten Feldern!) das Zeichen des aktuellen Spielers gesetzt wird. Diese Möglichkeit sollte unterbunden werden. Des Weiteren wäre es erwünscht, den Gewinner des Spiels auf irgendeine Weise hervorzuheben. Beide Anpassungen werden im nächsten Schritt angegangen.

Als erstes sollen bereits angeklickte und somit beschriftete Felder nicht erneut geändert werden können. Dies kann durch das Button-Attribut *disabled* erreicht werden. In der vorhandenen Implementierung des Spiels müsste *button_gedrueckt*, aber auch die Methode *neues_spiel* erweitert werden, sodass bei jedem Buttonklick das jeweilige Feld gesperrt wird und beim Starten eines neuen Spiels alle Felder wieder entsperrt werden.Gleich im Anschluss ist die Anpassung zu testen, damit immer kleine Implementierungsschritte und Testphasen beim Programmieren angewöhnt werden(Abbildung 23).

Abbildung 23: Tic-tac-toe App - Entwurf 2

Wenn beim Testen der bislang entworfenen App keine weiteren funktionsbezogenen Mängel festgestellt wurden, kann mit der Implementierung der Gewinnerermittlung begonnen werden. Die Prüfungslogik sollte wegen Übersichtlichkeit in eine eigene Methode ausgelagert werden, die nach jedem Zug aufgerufen wird um zu prüfen, ob der Gewinner schon feststeht. Wenn das der Fall ist, könnte das beispielsweise durch farbliche Kennzeichnung und eine größere Schrift hervorgehoben werden. Eine Diskussion im Unterricht bez. eines entsprechenden Algorithmus zur Umsetzung sollte erfolgen.

Die Implementierung der *check_gewonnen* Methode könnte in etwa wie folgt ausschauen:

```
def check_gewonnen(self):
                              #Positionen von moeglichen Gewinkombinationen
      Gewinnkombinationen = ( (0, 1, 2),  #Zeile 1
                              (3, 4, 5),  #Zeile 2
                              (6, 7, 8),  #Zeile 3
                              (0, 3, 6),  #Spalte 1
                              (1, 4, 7),  #Spalte 2
                              (2, 5, 8),  #Spalte 3
                              (0, 4, 8),  #quer:links oben->rechts unten
                              (2, 4, 6)  )#quer:links unten->rechts oben

      for kombination in gewinnkombinationen:

            button_a = 'btn' + str(kombination[0])
            button_b = 'btn' + str(kombination[1])
            button_c = 'btn' + str(kombination[2])

            if (self.root.ids[button_a].text == \
                  self.root.ids[button_b].text == \
                  self.root.ids[button_c].text == \
                  self.aktueller_spieler):

                        self.root.ids[button_a].font_size = 100
                        self.root.ids[button_a].background_color =(0,1,0,1)#gruen

                        self.root.ids[button_b].font_size = 100
                        self.root.ids[button_b].background_color =(0,1,0,1)#gruen

                        self.root.ids[button_c].font_size = 100
                        self.root.ids[button_c].background_color =(0,1,0,1)#gruen
```

Die App dürfte nun die gewünschte Funktionalität der Gewinnerhervorhebung aufweisen und visuell ansprechender wirken. Beim Testen dürfte jedoch auffallen, dass im Falle verbleibender Felder, nachdem das Spiel bereits entschieden wurde, die restlichen Felder immer noch angeklickt werden können. Dies müsste unterbunden werden. Dies könnte man durch eine ausgelagerte Methode erreichen, die bei einer treffenden Gewinnkombination aufgerufen wird, z.B.:

```
def spiel_beenden(self,gewinnkombination):
      for i in range (0,9):
            b = 'btn' + str(i)
            if i in gewinnkombination:
                  self.root.ids[b].font_size = 100
                  self.root.ids[b].background_color = (0,1,0,1) #gruen
            else:
                  self.root.ids[b].disabled = True
```

Die Implementierung sieht in der GUI dann wie in Abbildung 24 aus.

Abbildung 24: Tic-tac-toe App - Entwurf 3

Durch Tests oder Besprechung mit Studenten sollte zum Schluss gekommen werden, dass wieder die Methode *neues_spiel* anzupassen ist, damit bei Neubeginn die Schriftgröße und Buttonfarbe auf die Standardwerte gesetzt werden.

Im nächsten Schritt könnte als Erweiterung der App der Score eingeführt werden. Dies kann mit einer entsprechenden Methode und zwei zusätzlichen Klassenattributen, in denen der Punktestand gespeichert wird, erreicht werden. Optional kann die Farbe des Spielers mit dem höheren Punktestand farblich hervorgehoben werden. Letztendlich könnte die fertige App wie in Abbildung 25 ausschauen.

Abbildung 25: Tic-tac-toe App - Basisversion

6.13 Wiederholung und Vertiefung der Konzepte an einer Spiel-App (2)

Als Zusatzaufgabe könnte die Tic-tac-toe App um das Speielen 'gegen das Smartphone' erweitert werden. Es sollte so umgesetzt werden, dass ein umschalten zwischen Spieler- und Smartphonemodus ermöglicht wird. Beim Spiel gegen das Smartphone wird nach einer bestimmten Zeit (ca. 1-2 Sekunden) nach dem eigenen Zug eines der übrigen freien Felder nach Zufallsprinzip gewählt und mit dem Zeichen des Gegners (Smartphones) belegt. Der restliche Code der App bleibt erhalten, bis auf entsprechende Anpassungen, die durch die Erweiterung bedingt sind.

Im Rahmen der Implementierung würden insbesondere die Modulkonzepte wiederholt (Verwendung der API für die zeitliche Verzögerung eines Funktionsaufrufes, sowie Zufallsauswahl aus einer Sequenz) und problemlösendes Denken im Rahmen des Softwareentwicklungsprozesses gefördert.

Die fertige App mit der Erweiterung um den Smartphone-Spielmodus könnte z.B. wie folgt realisiert werden(Abbildung 26):

Abbildung 26: Tic-tac-toe App - erweiterte Version

Der Code zur fertigen Tic-tac-toe App kann Anhang 8.2.6 entnommen werden. Das Spiel gegen das Smartphone wird in der einfachen Version so realisiert, dass freie Felder willkürlich vom Smartphone gewählt werden. Als Fleißaufgabe für Fortgeschrittene könnte die Implementierung so umgesetzt werden, dass die Belegung der Felder nach einem bestimmter Algorithmus erfolgt, der die Gewinnchancen maximiert.

6.14 Evaluation

Im Rahmen von Kapitel 6 wurde ein Prototyp eines Curriculums für Programmieranfänger konzipiert und durch Vorschläge entsprechender Unterrichtsszenarien entfaltet. Die behandelten Konzepte für den Einstieg wurden aufgrund von Empfehlungen aus einschlägiger Literatur (Armstrong et al., 2013; Lahtinen et al., 2005; Milne & Rowe, 2002; Schulte & Bennedsen, 2006) und dem spezifischen App-Ansatz auf folgende festgelegt: Syntax, Variablen, grundlegende Datentypen, Bedingungen, Iterationen, Funktionen, GUI, Klassen, Methoden sowie Grundlagen von API und built-in Funktionen. Die Reihenfolge, in dem die Vermittlung der Konzepte erfolgte, basierte zum einen auf den Vorgaben des OO-Paradigma und zum anderen auf deren Schwierigkeitsgrad (Lahtinen et al., 2005). Einfachere Konzepte wurden vorgezogen.

Beim Programmiereinstieg durch App-Entwicklung handelt es sich um einen objektorientierten Ansatz, bei dem gleich zu Beginn mehrere zusammenhängende Konzepte vermittelt werden. Da das Curriculum jedoch für Personen mit einer höheren Einstiegsschwelle mit ausreichendem Abstraktionsvermögen vorgesehen ist, dürfte ein solcher Start kein Hindernis darstellen. Einerseits basiert das Verständnis des Objektbegriffs und dessen Charakteristika wie Eigenschaften und Funktionalitäten auf alltäglicher Realität und ist somit bis zu einem gewissen Grad schon vorhanden. Andererseits wird eine Programmiersprache wie eine natürliche Sprache gelernt – indem man Code ausgesetzt ist und sich mit ihm auseinandersetzt – auch wenn man ihn anfangs womöglich nicht vollständig versteht. Die Grundrisse und Zusammenhänge sind schon einigermaßen erkennbar, denn einfache Regeln jeglicher Syntax bzw. simple Algorithmen sind bereits intuitiv verständlich (Proulx et al., 2002). Deswegen wurden auch möglichst einfache aber sinnvolle Anfangsbeispiele verwendet, um gleich zu Beginn das Grundmotiv eines jeden Programms darzustellen – eine Funktionalität zu liefern.

Hinsichtlich der didaktischen Abfolge von Tätigkeiten wurden Richtlinien von Kölling und Rosenberg (2001) verwendet. Zu Beginn wird der Code nicht geschrieben, sondern zur Verfügung gestellt und gelesen – wie beim Erlernen mit einer neuen Fremdsprache. Das generelle Verständnis von Syntax Klassen, Methoden und deren Aufrufen wird dadurch vermittelt. Im Nachhinein werden kleine Änderungen am bestehenden Code durchgeführt, die anfangs von trivialen String- oder Variablenänderungen bis hin zu GUI Anpassungen und anspruchsvolleren Stringmanipulationen reichen. Der Grundgedanke dahinter ist, dass sich Studierende mit der grundlegenden Syntax und Semantik vertraut machen und einfache Anweisungen schreiben können. Des Weiteren lernen sie den Basiszyklus der Softwareentwicklung, der iterativ beim Programmieren angewendet wird: Codieren –

Ausführen – Testen – Fehlerbehebung. Im nächsten Schritt werden funktionale Änderungen einbezogen, indem vorhandene Methoden angepasst oder neue implementiert werden, die ein gewünschtes Verhalten der App realisieren. Allmählich werden die Kenntnisse so weit vertieft, bis Studierende eine bestimmte Funktionalität algorithmisch konzipieren und syntaktisch korrekt ausdrücken können. Dadurch sollten sie befähigt werden, einfache Apps weitestgehend selbst zu gestalten. Dies ist auch der Folgeschritt – ein relativ früher Beginn, einfache Anwendungsapps mit praktischem Nutzen zu codieren, um das Gefühl vermittelt zu bekommen, bereits effektiv programmieren zu können und schon erste Resultate in Form von fertigen Apps zu schöpfen.

In Bezug auf den Aufbau von Unterrichtseinheiten wird viel Wert auf Praxis gelegt. Das Programmieren vor den Studenten wird als sehr wichtig für bessere Lerneffekte hervorgehoben (Rubin, 2013). In der Regel werden Programmierkonzepte kurz theoretisch erläutert und vorgezeigt, um sie daraufhin in der Entwicklung von einfachen Apps, die auch im Alltag Verwendung finden können, praktisch anzuwenden. Die meisten Konzepte werden mehrmals in verschiedenen Apps verwendet, um auf diese Weise deren Anwendungsmöglichkeiten an unterschiedlichen Beispielen zu verdeutlichen und sie durch Wiederholung zu befestigen. Die fertigen Apps sind visuell und vom Umfang her natürlich nicht im Rang der am Markt verfügbaren, enthalten jedoch die jeweiligen Basisfunktionalitäten, mit denen ein realitätsnahes Programmieren ermöglicht wird. Die Erstellung von einfachen Apps als 'greifbaren' Endprodukten mit visualisierenden GUIs, sowie deren stetige Übertragung, Testen und Verwendung auf mobilen Geräten wurde erwägt, um dadurch die Motivation und Interesse am Programmieren bei Novizen zu steigern und sie für das Fach und eventuelle Vertiefung darin zu begeistern.

7 Zentrales Ergebnis der Arbeit

Das zentrale Ergebnis dieser Diplomarbeit ist das vorliegende Curriculum mit Unterrichtsszenarien, um den Einstieg ins Programmieren einfach und motivierend durch App-Entwicklung zu gestalten. Es eignet sich für Novizen mit einer höheren Einstiegsschwelle (ab ca. 14 Jahren) und ist breit angesetzt, sodass es in unterschiedlichen Lehrinstitutionen zumindest als Basis eingesetzt werden kann. Folgende Konzepte des Programmierens werden durch das Curriculum abgedeckt:

- Syntax/Semantik
- Klassen, Methoden
- Variablen, Ausdrücke, Zuweisungen
- grundlegende Datentypen
- GUI/Interaktion
- Bedingungen
- Funktionen, Parameter
- Iterationen
- API und built-in Funktionen (Basiskenntnisse)

Daneben werden Grundzüge der Softwareentwicklung vermittelt, sowie die algorithmische Denkweise gefördert. In den Unterrichtsszenarien wird Wert darauf gelegt, realitätsnahe Praxisbeispiele zu verwenden und einfache Anwendungsapps zu entwickeln, um relativ früh 'greifbare' Ergebnisse zu erzielen und Novizen in dieser Hinsicht zu begeistern. Der Vorzeigeeffekt einer fertigen App, die von Studenten selbst erstellt wurde, in Kombination mit der Verwendung von Smartphones oder anderen mobilen Geräten als neuen Technologien, dürfte lt. Studien sehr positive Motivationseffekte hervorrufen.

Eine Evaluierung des 'Apps-first' Curriculums im praktischen Umfeld, um zu zeigen, ob eine Einführung ins Programmieren durch App-Entwicklung tatsächlich die Motivation der Studenten erhöht, würde den Rahmen dieser Diplomarbeit sprengen. Daher kann dies für eine Folgestudie in Ausblick gestellt werden.

8 Anhang

8.1 Apps am mobilen Gerät starten

8.1.1 Ohne Installation

Für **Android** Systeme ist die einfachste und schnellste Methode in Python-Kivy erstellte Apps am mobilen Gerät laufen zu lassen, über den **Kivy Launcher**. Dies ist selbst eine App, in Rahmen welcher Kivy-Sourcecode ohne Installation ausgeführt werden kann. Sie kann über den Google Play Store installiert werden (https://play.google.com/store/apps/details?id=org.kivy.pygame).

Alternativ kann die .apk auf der Kivy Homepage herunterladen werden (http://kivy.org/#download)

Nachdem der Kivy Launcher am Smartphone installiert wurde, muss ein Verzeichnis mit dem Namen *kivy* angelegt werden, in dem wiederum Unterverzeichnisse mit dem Sourcecode von Apps angelegt bzw. kopiert werden. Der genaue Pfad muss wie folgt lauten:

/sdcard/kivy/<Appverzeichnis> z.B. /sdcard/kivy/Eingabemaske

Im Appverzeichnis müssen sich jeweils folgende Dateien befinden:

- main.py
- <appname>.kv (z.B. eingabemaske.kv)
- android.txt

Die android.txt Dateien muss folgende Informationen beinhalten:

title = Applikationsbezeichnung (z.B. title = KivyEingabemaske)

author = Name des Authors (z.B. author = Eldin D.)

orientation = <portrait|landscape> (z.B, orientation = portrait)

Wenn alles der Anleitung nach eingerichtet wurde, muss nur noch der Kivy Launcher gestartet werden. Vorhandene Apps, als Projekte dargestellt, sollten angezeigt werden. Einfach das gewünschte wählen um es auszuführen.

Die Übertragung von Dateien ist per Kabel möglich, oder über das WLAN – Vorschlag: Airdroid App - https://www.airdroid.com/

8.1.2 Mit Installation

Unter http://kivy.org/docs/guide/packaging.html gibt es ausführliche Anleitungen zum Erstellen von Installationsdateien aus App-Sourcecode, die betriebssystemspezifisch sind.

Für Android Betriebssysteme ist die Erstellung von Installationsdateien derzeit nur unter Linux möglich. Kivy stellt eine Linux Ubuntu Virtuelle Maschine zur Verfügung, die alle dafür benötigten Tools bereits enthält.

8.2 Verwendeter Code

8.2.1 Eingabemaske

eingabemaske.kv

```
BoxLayout:
	orientation: 'vertical'
	TextInput:
		id: eingabefeld1
		text:
	Button:
		id: button1
		text: 'eingeben'
		on_press: app.enter()
	Button:
		id: button2
		text: 'löschen'
		on_press: app.clear()
	Label:
		id: ausgabefeld1
		text:
```

Eingabemaske.py

```
from kivy.app import App

class Eingabemaske(App):

	def enter(self):
		a = self.root.ids.eingabefeld1.text
		self.root.ids.ausgabefeld1.text = a

	def clear(self):
		self.root.ids.eingabefeld1.text = ''
		self.root.ids.ausgabefeld1.text = ''
```

8.2.2 Währungskonverter

Waehrungskonverter.kv:

```
BoxLayout:
      orientation:'vertical'
      BoxLayout:
            orientation:'vertical'
            Label:
                  text: 'Währungskonverter'
                  font_size: 30
            Label:
                  #Platzhalter
            BoxLayout:
                  TextInput:
                        id: anzeige_betrag_eingabe
                        input_filter: 'float'
                        hint_text: 'Betrag eingeben..'
                        font_size: 30
                        bold: True
                        background_color: (0,1,0,1)
                  TextInput:
                        id: anzeige_ausgangswaehrung
                        text: app.ausgangswaehrung
                        font_size: 30
                        bold: True
                        disabled: True
                  TextInput:
                        text: ' = '
                        font_size: 30
                        bold: True
                        disabled: True
                  TextInput:
                        id: anzeige_betrag_ausgabe
                        font_size: 30
                        bold: True
                        disabled: True
                  TextInput:
                        id: anzeige_zielwaehrung
                        text: app.zielwaehrung
                        font_size: 30
                        bold: True
                        disabled: True
            Button:
                  text: 'konvertieren!'
                  font_size: 40
                  bold: True
                  color: (0.5,0,0,1)
                  on_press: app.konvertieren()
            Label: #Platzhalter
            Label:
                  id: kurslabel
                  text:'Aktueller Kurs:'
                  font_size: 30
                  color: (0,1,0,1)
```

```
            BoxLayout:
                  TextInput:
                        text: '1'
                        font_size: 30
                        disabled: True
                        background_color: (0,50,0,0.3)
                  TextInput:
                        id: aktuell_ausgangswaehrung
                        text: app.ausgangswaehrung
                        font_size: 30
                        disabled: True
                        background_color: (0,50,0,0.3)
                  TextInput:
                        text: ' = '
                        font_size: 30
                        disabled: True
                        background_color: (0,50,0,0.3)
                  TextInput:
                        id: aktuell_kurs
                        text: str(app.kurs)
                        font_size: 30
                        input_filter: 'float'
                        disabled: True
                        background_color: (0,50,0,0.3)
                  TextInput:
                        id: aktuell_zielwaehrung
                        text: app.zielwaehrung
                        font_size: 30
                        disabled: True
                        background_color: (0,50,0,0.3)
            BoxLayout:
                  Button:
                        size_hint:.4,1
                        text: 'entsperren'
                        #text_size: self.size
                        font_size: 20
                        on_press: app.kurs_entsperren()
                  Button:
                        size_hint:.6,1
                        text: 'Änderung übernehmen'
                        font_size: 20
                        on_press: app.kurs_aendern()
      BoxLayout:
```

Waehrungskonverter.py

```
from kivy.app import App

class Waehrungskonverter(App):

      ausgangswaehrung = '€'
      zielwaehrung = '$'
```

```
kurs = 1.3

def konvertieren(self):

      if self.root.ids.anzeige_betrag_eingabe.text == '':
            #self.root.ids.anzeige_betrag_eingabe.hint_text:
            self.root.ids.anzeige_betrag_ausgabe.text = ''
            #wenn eingabe gelöscht wird
            return
      input = self.root.ids.anzeige_betrag_eingabe.text

      betrag_eingabe = float(input)
      betrag_konvertiert = betrag_eingabe * self.kurs
      betrag_konv_gerundet = round(betrag_konvertiert,2)

      self.root.ids.anzeige_betrag_ausgabe.text = str(betrag_konv_gerundet)



def kurs_aendern(self):
      if self.root.ids.aktuell_ausgangswaehrung.text == '' or \
            self.root.ids.aktuell_kurs.text == '' or\
            self.root.ids.aktuell_zielwaehrung.text == '' or\
            self.root.ids.aktuell_ausgangswaehrung.text == \
            self.root.ids.aktuell_zielwaehrung.text:

            self.root.ids.kurslabel.text = 'Fehler in Kurseingabe!'
            self.root.ids.anzeige_betrag_eingabe.disabled = True
            return
      else:
            self.ausgangswaehrung =\
                  self.root.ids.aktuell_ausgangswaehrung.text
            self.kurs = float(self.root.ids.aktuell_kurs.text)
            self.zielwaehrung = self.root.ids.aktuell_zielwaehrung.text

            self.anzeige_aktualisieren()

def anzeige_aktualisieren(self):
      self.root.ids.anzeige_ausgangswaehrung.text = self.ausgangswaehrung
      self.root.ids.anzeige_zielwaehrung.text = self.zielwaehrung
      self.root.ids.aktuell_ausgangswaehrung.disabled = True
      self.root.ids.aktuell_kurs.disabled = True
      self.root.ids.aktuell_zielwaehrung.disabled = True

      if self.root.ids.anzeige_betrag_eingabe.text != '':
                  self.konvertieren()
      if self.root.ids.anzeige_betrag_eingabe.disabled == True:
                  self.root.ids.anzeige_betrag_eingabe.disabled = False
      if self.root.ids.kurslabel.text == 'Fehler in Kurseingabe!':
            self.root.ids.kurslabel.text = 'Aktueller Kurs:'

def kurs_entsperren(self):
      self.root.ids.aktuell_ausgangswaehrung.disabled = False
      self.root.ids.aktuell_kurs.disabled = False
      self.root.ids.aktuell_zielwaehrung.disabled = False
```

8.2.3 Einkaufsliste-App

einkaufsliste.kv

```
BoxLayout:
	orientation: 'vertical'
	BoxLayout:
		orientation: 'vertical'
		TextInput:
			id: eingabefeld1
			font_size: 30
		Button:
			id: button1
			text: 'Produkt zu Liste hinzufügen'
			background_color: (0,1,0,1)
			on_press: app.enter_item()
		Button:
			id: button2
			text: 'Produkt an Stelle X aus Liste löschen'
			background_color: (1,0,0,1)
			on_press: app.delete_item()
		Button:
			id: button3
			text: 'Liste leeren'
			font_size: 20
			background_color: (1,0,0,1)
			bold: True
			on_press: app.delete_list()
	BoxLayout:
		orientation: 'vertical'
		Label:
			id: ausgabefeld1
			font_size: 20
			text_size: self.size
			halign: 'center'
			valign: 'middle'
```

Einkaufsliste.py

```
from kivy.app import App

class Einkaufsliste(App):

	einkaufskorb = []

	def enter_item(self):
		new_item = self.root.ids.eingabefeld1.text
		if not self.vorhanden(new_item):
			self.einkaufskorb.append(new_item)
			self.root.ids.ausgabefeld1.text = str(self.einkaufskorb)
		self.root.ids.eingabefeld1.text = ''

	def delete_item(self):
		position = int(self.root.ids.eingabefeld1.text) - 1
```

```
            #catch no entry
            if abs(position) >= len(self.einkaufskorb):
                  return
            else:
                  del self.einkaufskorb[position]
                  self.root.ids.ausgabefeld1.text = str(self.einkaufskorb)


      def delete_list(self):
            if len(self.einkaufskorb)>0:
                  del self.einkaufskorb[:]
                  self.root.ids.ausgabefeld1.text = str(self.einkaufskorb)
```

8.2.4 BMI (Body Mass Index) Calculator

bmicalculator.kv

```
BoxLayout:
      orientation:'vertical'
      Label:
            size_hint: 1,.125
      Label:
            text: 'BMI Rechner'
            font_size: 30
            size_hint: 1,.125
      Label:
            size_hint: 1,.125
      BoxLayout:
            orientation:'vertical'
            BoxLayout:
                  Label:
                        text: 'Alter (in Jahren)'
                        font_size: 20
                  TextInput:
                        id: alter_jahre
                        input_filter: 'int'
                        font_size: 30
                        bold: True
                        background_color: (100,200,200,.7)
            BoxLayout:
                  Label:
                        text: 'Größe (in cm)'
                        font_size: 20
                  TextInput:
                        id: groesse_cm
                        input_filter: 'float'
                        font_size: 30
                        bold: True
                        background_color: (100,200,200,.7)
            BoxLayout:
                  Label:
                        text: 'Gewicht (in kg)'
                        font_size: 20
                  TextInput:
```

```
                id: gewicht_kg
                input_filter: 'float'
                font_size: 30
                bold: True
                background_color: (100,200,200,.7)
      BoxLayout:
            Label:
                text: 'Geschlecht (m/w)'
                font_size: 20
            TextInput:
                id: geschlecht_m_w
                font_size: 30
                bold: True
                background_color: (100,200,200,.7)
BoxLayout:
      orientation:'vertical'
      Label:
            text: ''
            id: meldung
      BoxLayout:
            Button:
                text: 'reset'
                size_hint: .33,1
                font_size: 40
                bold: True
                color: (0.5,0,0,1)
                on_press: app.reset()
            Button:
                text: 'Werte ermitteln'
                font_size: 40
                bold: True
                color: (0.5,0,0,1)
                on_press: app.werte_ermitteln()
GridLayout:
      cols:3
      Label:
            text: 'BMI'
            font_size: 20
      Label:
            text: 'Ideales Gewicht'
            font_size: 20
      Label:
            text: 'Fettanteil'
            font_size: 20
      TextInput:
            id: bmi
            text: ''
            font_size: 30
            background_color: (0,200,200,.7)
            disabled: 'true'
      TextInput:
            id:ideal_weight
            text: ''
            font_size: 30
            background_color: (0,200,200,.7)
```

```
            disabled: 'true'
        TextInput:
            id:bfp
            text: ''
            font_size: 30
            background_color: (0,1,0,.7)
            disabled: 'true'
```

BMICalculator.py

```
from kivy.app import App

class BMICalculator(App):

    def werte_ermitteln(self):
        if self.root.ids.meldung.text != '':
            self.root.ids.meldung.text = ''
        try:
            alter = int(self.root.ids.alter_jahre.text)
            groesse = float(self.root.ids.groesse_cm.text)
            gewicht = float(self.root.ids.gewicht_kg.text)
        except:
            self.root.ids.meldung.text = 'Eingabefehler!'
            return

        geschlecht = self.root.ids.geschlecht_m_w.text
        if geschlecht not in('m','w'):
            self.root.ids.meldung.text = 'Eingabefehler (Geschlecht)!'
            return

        my_bmi = self.bmi_berechung(groesse,gewicht)
        my_bmi_farbe = self.kolorieren('bmi',my_bmi)

        self.root.ids.bmi.text = str(round(my_bmi,2))
        self.root.ids.bmi.background_color = my_bmi_farbe

        my_bfp = self.bfp_berechnung(my_bmi,alter,geschlecht)
        my_bfp_farbe = self.kolorieren('bfp',my_bfp)
        self.root.ids.bfp.text = str(round(my_bfp,2)) + ' %'
        self.root.ids.bfp.background_color = my_bfp_farbe
        #ev. Meldungen ausgeben 'time for a run' etc.
        self.root.ids.ideal_weight.text = \
            self.idealgewicht_berechnung(groesse,geschlecht)


    def eingabe_reset(self):
        self.root.ids.alter_jahre.text = ''
        self.root.ids.groesse_cm.text = ''
        self.root.ids.gewicht_kg.text = ''
        self.root.ids.geschlecht_m_w.text = ''

    def ausgabe_reset(self):
        self.root.ids.bmi.text = ''
        self.root.ids.ideal_weight.text = ''
```

```
            self.root.ids.bfp.text = ''
            self.root.ids.meldung.text = ''
            #ev. farbe auf weiß setzen

      def reset(self):
            self.eingabe_reset()
            self.ausgabe_reset()

      def bmi_berechung(self,cm,kg):
            bmi = kg / ((cm/100)**2)
            return bmi

      def idealgewicht_berechnung(self,groesse,geschlecht):

            faktor_von = 19
            faktor_bis = 24

            if geschlecht == 'm':
                  faktor_von = 20
                  faktor_bis = 25

            idealgewicht_von = (groesse/100)**2 * faktor_von
            idealgewicht_bis = (groesse/100)**2 * faktor_bis

            idealgewicht_bereich  =  str(round(idealgewicht_von,1))  +  '  -  '  +
str(round(idealgewicht_bis,1)) + ' kg'
            return idealgewicht_bereich


      def bfp_berechnung(self,bmi,jahre,m_w):
            geschlecht_faktor = 0
            if m_w == 'm':
                  geschlecht_faktor = 1

            bfp = 1.2 * bmi + 0.23 * jahre - 10.8 * geschlecht_faktor -5.4
            #Children's formula to do
            return bfp

      def kolorieren(self,merkmal,wert):
            if merkmal == 'bmi':
                  if wert < 18.5:
                        return (1,1,0,1)
                  elif wert >=18.5 and wert < 25:
                        return (0,1,0,1)
                  else:
                        return (1,0,0,1)
            else:
                  return (0,1,0,1) #to do for bfp
```

8.2.5 Einfacher Taschenrechner

Taschenrechner.kv

```
BoxLayout:
      orientation: 'vertical'
      TextInput:
            id: display
            font_size: 35
            text:
            size_hint: 1,.25
      GridLayout:
            rows: 6
            cols: 4
            Button:
                  text: 'M+'
                  font_size: 30
                  bold: 'true'
                  background_color: (0,0,1,1)
                  on_press: app.mr_add()
            Button:
                  text: 'M-'
                  font_size: 30
                  bold: 'true'
                  background_color: (0,0,1,1)
                  on_press: app.mr_empty()
            Button:
                  text:'MR'
                  font_size: 30
                  bold: 'true'
                  background_color: (0,0,1,1)
                  on_press: app.mr_call()
            Button:
                  text:'C'
                  font_size: 30
                  bold: 'true'
                  background_color: (1,0,0,1)
                  on_press: app.clear()
            Button:
                  text: '('
                  font_size: 30
                  bold: 'true'
                  background_color: (0,1,0,1)
                  on_press: app.pressed('(')
            Button:
                  text: ')'
                  font_size: 30
                  bold: 'true'
                  background_color: (0,1,0,1)
                  on_press: app.pressed(')')
            Button:
                  text:'+/-'
                  font_size: 30
                  bold: 'true'
                  background_color: (0,1,0,1)
```

```
      on_press: app.invert_value()
Button:
      text:'%'
      font_size: 30
      bold: 'true'
      background_color: (0,1,0,1)
      on_press: app.percent()
Button:
      text: '7'
      font_size: 30
      bold: 'true'
      on_press: app.pressed('7')
Button:
      text: '8'
      font_size: 30
      bold: 'true'
      on_press: app.pressed('8')
Button:
      text:'9'
      font_size: 30
      bold: 'true'
      on_press: app.pressed('9')
Button:
      text:'/'
      font_size: 30
      bold: 'true'
      background_color: (0,1,0,1)
      on_press: app.pressed('/')
Button:
      text:'4'
      font_size: 30
      bold: 'true'
      on_press: app.pressed('4')
Button:
      text:'5'
      font_size: 30
      bold: 'true'
      on_press: app.pressed('5')
Button:
      text:'6'
      font_size: 30
      bold: 'true'
      on_press: app.pressed('6')
Button:
      text:'*'
      font_size: 30
      bold: 'true'
      on_press: app.pressed('*')
      background_color: (0,1,0,1)
Button:
      text:'1'
      font_size: 30
      bold: 'true'
      on_press: app.pressed('1')
Button:
```

```
            text:'2'
            font_size: 30
            bold: 'true'
            on_press: app.pressed('2')
      Button:
            text:'3'
            font_size: 30
            bold: 'true'
            on_press: app.pressed('3')
      Button:
            text:'-'
            font_size: 30
            bold: 'true'
            background_color: (0,1,0,1)
            on_press: app.pressed('-')
      Button:
            text:'0'
            font_size: 30
            bold: 'true'
            on_press: app.pressed('0')
      Button:
            text:'.'
            font_size: 30
            bold: 'true'
            on_press: app.pressed('.')
      Button:
            text:'='
            font_size: 30
            bold: 'true'
            background_color: (0,1,0,1)
            on_press: app.calculate()
      Button:
            text:'+'
            font_size: 30
            bold: 'true'
            background_color: (0,1,0,1)
            on_press: app.pressed('+')
```

Taschenrechner.py

```
from kivy.app import App

class Taschenrechner(App):

      mr = ''

      def pressed(self,char):

            anzeige_aktuell = self.root.ids.display.text
            anzeige_neu = ''

            if '=' in anzeige_aktuell:
                  ergebnis_index = anzeige_aktuell.find('=')+1
                  anzeige_neu = str(anzeige_aktuell[ergebnis_index:]) + char
            else:
```

```
            anzeige_neu = anzeige_aktuell + char

      self.root.ids.display.text = anzeige_neu

def calculate(self):
      anzeige_aktuell = self.root.ids.display.text
      if '=' in anzeige_aktuell:
            return
      else:
            result = eval(anzeige_aktuell)
            self.root.ids.display.text = anzeige_aktuell + '\n ='+
                                         str(result)

def index_of_result(self):
      anzeige = self.root.ids.display.text
      if '=' in anzeige:
            ergebnis_index = anzeige.find('=')+1
            return ergebnis_index
      else:
            return 0

def mr_add(self):
      self.mr = eval(self.root.ids.display.text)

def mr_empty(self):
      self.mr = ''

def mr_call(self):
      self.root.ids.display.text = self.root.ids.display.text + str(self.mr)

def clear(self):
      self.root.ids.display.text = ''

def invert_value(self):
      result = eval(self.root.ids.display.text)
      if float(result)>0:
            self.root.ids.display.text = '-' + str(result)
      else:
            self.root.ids.display.text = str(result * -1)

def percent(self):
      result = eval(self.root.ids.display.text)
      percentage = float(result) / 100
      self.root.ids.display.text = str(percentage)
```

8.2.6 TicTacToe

tictactoe.kv

```
BoxLayout:
	orientation: 'vertical'
	Button:
		size_hint: 1,.25
		text: 'neues Spiel starten!'
		font_size: 30
		on_press: app.neues_spiel()
	Button:
		id: btn_spielmodus
		size_hint: 1,.125
		text: 'gegen Smartphone spielen!'
		on_press: app.spielmodus_aendern()
		font_size: 30
		text_size: self.size
		halign: 'center'
		valign: 'middle'
	BoxLayout:
		size_hint: 1,.125
		Button:
			id: btn_beginner_links
			text: 'X beginnt'
			font_size: 30
			on_press: app.setze_spieler('X')
		Button:
			id: btn_beginner_rechts
			text: 'O beginnt'
			font_size: 30
			on_press: app.setze_spieler('O')
	BoxLayout:
		size_hint: 1,.125
		Label:
			id: lbl_punkte_x_beschriftung
			text: 'Punkte X:'
			font_size: 30
		Label:
			id: lbl_punkte_x
			text: '0'
			font_size: 30
	BoxLayout:
		size_hint: 1,.125
		Label:
			id: lbl_punkte_o_beschriftung
			text: 'Punkte O:'
			font_size: 30
		Label:
			id: lbl_punkte_o
			text: '0'
			font_size: 30
	GridLayout:
		rows: 3
		cols: 3
```

```
        Button:
            id: btn0
            font_size: 80
            on_press: app.button_gedrueckt('btn0')
        Button:
            id: btn1
            font_size: 80
            on_press: app.button_gedrueckt('btn1')
        Button:
            id: btn2
            font_size: 80
            on_press: app.button_gedrueckt('btn2')
        Button:
            id: btn3
            font_size: 80
            on_press: app.button_gedrueckt('btn3')
        Button:
            id: btn4
            font_size: 80
            on_press: app.button_gedrueckt('btn4')
        Button:
            id: btn5
            font_size: 80
            on_press: app.button_gedrueckt('btn5')
        Button:
            id: btn6
            font_size: 80
            on_press: app.button_gedrueckt('btn6')
        Button:
            id: btn7
            font_size: 80
            on_press: app.button_gedrueckt('btn7')
        Button:
            id: btn8
            font_size: 80
            on_press: app.button_gedrueckt('btn8')
```

Tictactoe.py

```
from kivy.app import App
from kivy.clock import Clock
import random

class Tictactoe(App):

    aktueller_spieler = 'X' #standardmäßig startet X
    spiel_gegen_smartphone = False
    score_X = 0
    score_O = 0

    def button_gedrueckt(self,id):
        self.root.ids[id].text = self.aktueller_spieler
```

```
        self.root.ids[id].disabled = True
        game_over = self.check_gewonnen()
        self.spieler_wechseln()

        if (self.spiel_gegen_smartphone and not game_over):
              Clock.schedule_once(self.smartphone_zug, 1.5)

  def spieler_wechseln(self):
        if self.aktueller_spieler == 'X':
              self.aktueller_spieler = 'O'
        else:
              self.aktueller_spieler = 'X'

  def neues_spiel(self):
        for i in range(0,9):
              b = 'btn' + str(i)
              self.root.ids[b].text = ''
              self.root.ids[b].disabled = False
              self.root.ids[b].font_size = 80
              self.root.ids[b].background_color = (1,1,1,1)

  def setze_spieler(self, zeichen):
        self.aktueller_spieler = zeichen
        if self.spiel_gegen_smartphone and self.aktueller_spieler == 'O':
              Clock.schedule_once(self.smartphone_zug, 1.5)


  def check_gewonnen(self):

        gewinnkombinationen = ( #Positionen von moeglichen Gewinkombinationen
                                (0, 1, 2),  #Zeile 1
                                (3, 4, 5),  #Zeile 2
                                (6, 7, 8),  #Zeile 3
                                (0, 3, 6),  #Spalte 1
                                (1, 4, 7),  #Spalte 2
                                (2, 5, 8),  #Spalte 3
                                (0, 4, 8),  #quer: links oben ->rechts unten
                                (2, 4, 6)   #quer: links unten ->rechts oben
                                                        )
        for kombination in gewinnkombinationen:

              button_a = 'btn' + str(kombination[0])
              button_b = 'btn' + str(kombination[1])
              button_c = 'btn' + str(kombination[2])

              if (self.root.ids[button_a].text == \
                    self.root.ids[button_b].text == \
                    self.root.ids[button_c].text == \
                    self.aktueller_spieler):
                          self.spiel_beenden(kombination)
                          return True
        return False
```

```
def spiel_beenden(self,gewinnkombination):
      for i in range (0,9):
            b = 'btn' + str(i)
            if i in gewinnkombination:
                  self.root.ids[b].font_size = 100
                  self.root.ids[b].background_color = (0,255,0,1) #gruen
            else:
                  self.root.ids[b].disabled = True
      self.score_aktualisieren()

def score_aktualisieren(self):
      if self.aktueller_spieler == 'X':
            self.score_X = self.score_X + 1
            self.root.ids.lbl_punkte_x.text=str(self.score_X)
      else:
            self.score_O = self.score_O + 1
            self.root.ids.lbl_punkte_o.text=str(self.score_O)

def spielmodus_aendern(self):
      if self.spiel_gegen_smartphone:
            self.spiel_gegen_smartphone = False
            self.root.ids.btn_spielmodus.text = 'gegen Smartphone spielen!'
            self.root.ids.btn_spielmodus.background_color = (1,1,1,1)
            self.root.ids.btn_beginner_links.text = 'X beginnt!'
            self.root.ids.lbl_punkte_x_beschriftung.text = 'Punkte X:'
            self.root.ids.btn_beginner_rechts.text = 'O beginnt!'
            self.root.ids.lbl_punkte_o_beschriftung.text = 'Punkte O:'
      else:
            self.spiel_gegen_smartphone = True
            self.root.ids.btn_spielmodus.text=\
            'Spiel gegen Smartphone(du bistX)!-klicken um Modus zu ändern..'
            self.root.ids.btn_spielmodus.background_color = (1,0,0,1)
            self.root.ids.btn_beginner_links.text = 'Ich beginne!'
            self.root.ids.lbl_punkte_x_beschriftung.text = 'Meine Punkte:'
            self.root.ids.btn_beginner_rechts.text = 'Smartphone beginnt!'
            self.root.ids.lbl_punkte_o_beschriftung.text='SmartphonePunkte'


      self.score_X = 0
      self.root.ids.lbl_punkte_x.text=str(self.score_X)
      self.score_O = 0
      self.root.ids.lbl_punkte_o.text=str(self.score_O)

      self.neues_spiel()

def smartphone_zug(self,dt):

      freie_felder = []

      for i in range(0,9):
            if not self.root.ids['btn'+str(i)].disabled: # not in ('X','O'):
                  freie_felder.append(i)
```

```
if len(freie_felder) == 0:
      return
else:
      zufallszahl = random.choice(freie_felder)
      self.root.ids['btn'+str(zufallszahl)].text=\
            self.aktueller_spieler
      self.root.ids['btn'+str(zufallszahl)].disabled = True
      self.check_gewonnen()
      self.spieler_wechseln()
```

9 Abbildungsverzeichnis

Abbildungsverzeichnis

10 Tabellenverzeichnis

Tabellenverzeichnis

11 Literaturverzeichnis

Argent, L., Depper, B., Fajardo, R., Gjertson, S., Leutenegger, S. T., Lopez, M. A., & Rutenbeck, J. (2006). Building a game development program. *Computer*, *39*(6), 52–60.

Armstrong, C., Asanovic, K., & Babiceanu, R. F. (2013). *Computer Science Curricula 2013*. USA: ACM and the IEEE Computer Society.

Barnes, D. J., & Kölling, M. (2009). *Java lernen mit BlueJ: eine Einführung in die objektorientierte Programmierung*. Deutschland Pearson

Becker, K. (2001). Teaching with games: the minesweeper and asteroids experience. *Journal of Computing Sciences in Colleges*, *17*(2), 23–33.

Chen, Z., & Marx, D. (2005). Experiences with Eclipse IDE in programming courses. *Journal of Computing Sciences in Colleges*, *21*(2), 104–112.

Ciampa, K. (2014). Learning in a mobile age: an investigation of student motivation. *Journal of Computer Assisted Learning*, *30*(1), 82–96.

Computing Curricula, (2001). *Computer Science, Final Report, December 2001*. The Joint Task Force on Computing Curricula, IEEE Computer Society, Association for Computing Machinery

Davies, S., Polack-Wahl, J. A., & Anewalt, K. (2011). A snapshot of current practices in teaching the introductory programming sequence. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (pp. 625–630). USA, New York: ACM.

Deimann, M. (2002). Motivationale Bedingungen beim Lernen mit Neuen Medien. In W.-G. Bleek, D. Krause, H. Oberquelle, & B. Pape (Eds.), *Medienunterstütztes Lernen - Beiträge von der WissPro-Wintertagung 2002.* (pp. 61–70). Hamburg: Universität Hamburg.

Denny, P., Luxton-Reilly, A., Tempero, E., & Hendrickx, J. (2011). Understanding the syntax barrier for novices. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 208–212). USA, New York: ACM.

Futschek, G. (2006). Algorithmic thinking: the key for understanding computer science. In *Lecture Notes in Computer Science 4226*, Springer, pp. 159 - 168.

Goldwasser, M. H., & Letscher, D. (2008). Using Python To Teach Object-Oriented Programming in CS1. *Innov. Technol. Comput. Sci. Ed.(June 2008)*.

Holvikivi, J. (2010). Conditions for successful learning of programming skills. In *Key Competencies in the Knowledge Society*, *324*, 155–164. Springer.

Holz, J., Leonhardt, T., & Schroeder, U. (2011). Using smartphones to motivate secondary school students for informatics. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research* (pp. 89–94). USA, New York: ACM.

Joosten, T. (2010). Mobile learning and social media: Increasing engagement and interactivity. Paper presented at the New Media Consortium Conference, June 9-12, 2010. Anaheim, CA.

Kasurinen, J., & Nikula, U. (2007). Lower dropout rates and better grades through revised course infrastructure. In *Proceedings of the 10th International Conference on Computers and Advanced Technology in Education* (pp. 152–157).

Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. *Communications of the ACM*, *50*(7), 58–64.

Keller, J. M. (1983). Motivational design of instruction. *Instructional Design Theories and Models: An Overview of Their Current Status*, *1*, 383–434.

Kim, Y. H., Kim, D. J., & Wachter, K. (2013). A study of mobile user engagement (MoEN): Engagement motivations, perceived value, satisfaction, and continued engagement intention. *Decision Support Systems*, *56*, 361–370.

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Computer Science Education*, *13*(4), 249–268.

Kölling, M., & Rosenberg, J. (2001). Guidelines for teaching object orientation with Java. *ACM SIGCSE Bulletin,33*(3), 33–36.

Koulouri, T., Lauria, S., & Macredie, R. D. (2014). Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 26.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin, 37*, 14–18.

Leutenegger, S., & Edgington, J. (2007). A games first approach to teaching introductory programming. *ACM SIGCSE Bulletin*, *39*(1), 115–118.

Linn, M. C., & Dalbey, J. (1989). Cognitive consequences of programming instruction. In E. Soloway & J. C. Sphorer (Eds.), *Studying the Novice Programmer* (pp. 57–81). Hillsdale, NJ: Lawrence Erlbaum Associates.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., et al. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, *36*(4), 119–150.

Lorenzen, T., Heilman, W. (2002). CS1 and CS2: write computer games in Java! SIGCSE Bulletin, 34(4), 99-100.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., et al. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, *33*(4), 125–180.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239–264.

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—views of students and tutors. *Education and Information Technologies*, *7*(1), 55–66.

Mueller, F., & Hosking, A. L. (2003). Penumbra: an Eclipse plugin for introductory programming. In

Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange (pp. 65–68). USA, New York: ACM.

Nienaltowski, M.-H., Pedroni, M., & Meyer, B. (2008). Compiler error messages: What can help novices? *ACM SIGCSE Bulletin*, *40*(1), 168–172.

Patterson-McNeill, H. (2006). Experience: from C++ to Python in 3 easy steps. *Journal of Computing Sciences in Colleges*, *22*(2), 92–96.

Peres S. (June 2, 2014). "iTunes App Store Now Has 1.2 Million Apps, Has Seen 75 Billion Downloads To Date", http://techcrunch.com/2014/06/02/itunes-app-store-now-has-1-2-million-apps-has-seen-75-billion-downloads-to-date/ (retreived: 06.08.2015)

Proulx, V. K., Raab, J., & Rasala, R. (2002). Objects from the beginning-with GUIs. *ACM SIGCSE Bulletin*, *34*(3), 65–69.

Radenski, A. (2006). Python First: A lab-based digital introduction to computer science. In *ACM SIGCSE Bulletin* (Vol. 38, pp. 197–201).

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … others. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60–67.

Rubin, M. J. (2013). The effectiveness of live-coding to teach introductory programming. In *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 651–656). USA, New York: ACM.

Schmalt, H.-D., & Heckhausen, H. (1992). Motivation. In H. Spada (Ed.), *Lehrbuch Allgemeine Psychologie*. Bern: Huber.

Schulte, C., & Bennedsen, J. (2006). What do teachers teach in introductory programming? In *Proceedings of the second international workshop on Computing education research* (pp. 17–28). USA, New York: ACM.

Schwabe, G., & Göth, C. (2005). Mobile learning with a mobile game: design and motivational effects. *Journal of Computer Assisted Learning*, *21*(3), 204–216.

Slany, W. (2012). A mobile visual programming system for Android smartphones and tablets. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on* (pp. 265–

266). Austria, Innsbruck: IEEE.

Storey, M.-A., Damian, D., Michaud, J., Myers, D., Mindel, M., German, D., … Hargreaves, E. (2003). Improving the usability of Eclipse for novice programmers. In *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange* (pp. 35–39). USA, New York: ACM.

Su, C.-H., & Cheng, C.-H. (2015). A mobile gamification learning system for improving the learning motivation and achievements. *Journal of Computer Assisted Learning*, *31*(3), 268–286.

Sweedyk, E., deLaet, M., Slattery, M. C., & Kuffner, J. (2005). Computer games and cs education: why and how. *ACM SIGCSE Bulletin, 37*, 256–257.

Van Roy, P., Armstrong, J., Flatt, M., & Magnusson, B. (2003). The role of language paradigms in teaching programming. *ACM SIGCSE Bulletin, 35*, 269–270.

Watson, C., & Li, F. W. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 39–44). USA, New York: ACM.

Yamamoto, N., & Wakahara, T. (2013). An Interactive Learning System Using Smartphone for Improving Students Learning Motivation. *Lecture Notes in Electrical Engineering, 253*, 305–310.